ACELERACIÓN DE ALGORITMOS PARA IMÁGENES HIPERESPECTRALES CON OPENCL

Rodríguez Navarro, Jose Miguel
Orueta Moreno, Carlos

GRADO EN INGENIERÍA DE COMPUTADORES. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado en Ingeniería de Computadores

Madrid, 16 de junio de 2016

Directores:

Bernabé García, Sergio
Botella Juan, Guillermo

Autorización de difusión

Rodríguez Navarro, Jose Miguel
Orueta Moreno, Carlos

Madrid, a 16 de junio de 2016

Los abajo firmantes, matriculados en el Grado de Ingeniería de Computadores de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "ACELERACIÓN DE ALGORITMOS PARA IMÁGENES HIPERESPECTRALES CON OPENCL", realizado durante el curso académico 2015-2016 bajo la dirección de Bernabé García, Sergio y la codirección de Botella Juan, Guillermo en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Agradecimientos

En primer lugar queremos agradecer a nuestros directores de proyecto, Sergio Bernabé García y Guillermo Botella Juan por darnos la oportunidad de trabajar en un proyecto tan innovador como el aquí presentado.

Un proyecto con muchas dificultades desde el comienzo pero que gracias a la perseverancia de nuestros directores hemos llevado a buen termino. No solo nos han proporcionado las herramientas adecuadas si no que además nos han guiado para encontrar las mejores soluciones. Con una buena comunicación y reuniones constantes.

A familiares y amigos por estar en los buenos y sobre todo en los malos momentos con su apoyo incondicional.

Agradecimientos personales

Carlos Orueta Moreno

Me gustaría agradecer a mis padres por apoyarme en todo momento y tener plena disposición a ayudarme en todo lo que pudieran.

A mi hermana y su pareja por hacerme ver los problemas de otra manera y darme la capacidad para poder afrontarlos.

A los amigos que durante estos años en la carrera nos hemos ayudado, apoyado, y por supuesto lo hemos pasado genial juntos. Un agradecimiento especial Marcos e Iván por todos los momentos que hemos vivido.

Por supuesto, agradecer a mi compañero por el esfuerzo tan duro que hemos tenido que realizar para poder hacer realidad este proyecto. Y a los directores que nos han guiado y ayudado siempre que les ha sido posible.

Por último me gustaría agradecer también a tres grandes amigas. Lucía, Mapi y María, que me han animado en los momentos más duros y me han estado apoyando siempre. Muchas gracias chicas.

Índice general

Índice de figuras

Índice de tablas

Resumen

En el presente trabajo se propone dar solución a uno de los problemas principales surgido en el campo del análisis de imágenes hiperespectrales. En las últimas décadas este campo está siendo muy activo, por lo que es de vital importancia tratar su problema principal: mezcla espectral. Muchos algoritmos han tratado de solucionar este problema, pero que a través de este trabajo se propone una cadena nueva de desmezclado en paralelo, para ser acelerados bajo el paradigma de programación paralela de OpenCl. Este paradigma nos aporta el modelo de programación unificada para acelerar algoritmos en sistemas heterogéneos.

Podemos dividir el proceso de desmezclado espectral en tres etapas. La primera tiene la tarea de encontrar el número de píxeles puros, llamaremos endmembers a los píxeles formados por una única firma espectral, utilizaremos el algoritmo conocido como *Geometry-based Estimation of number of endmembers*, **GENE**. La segunda etapa se encarga de identificar los píxel endmembers y extraerlos junto con todas sus bandas espectrales, para esta etapa se utilizará el algoritmo conocido por *Simplex Growing Algorithm*, **SGA**. En la última etapa se crean los mapas de abundancia para cada uno de los endmembers encontrados, de esta etapa será encargado el algoritmo conocido por, *Sum-to-one Constrained Linear Spectral Unmixing*, **SCLSU**.

Las plataformas utilizadas en este proyecto han sido tres: **CPU**, Intel Xeon E5-2695 v3, **GPU**, NVidia GeForce GTX 980, **Acelerador**, Intel Xeon Phi 31S1P. La idea de este proyecto se basa en realizar un análisis exhaustivo de los resultados obtenidos en las diferentes plataformas, con el fin de evaluar cuál se ajusta mejor a nuestras necesidades.

Palabras clave

Imágenes hiperespectrales, desmezclado espectral, computación paralela, OpenCl, sistemas heterogéneos.

Abstract

In this work it intends give a solution to a problem hiperespectral images analysis field. This field is growing due to the analysis of hyperspectral images. Many algorithms have tried to solve this problem, this paper bring new algorithms to be accelerated under the OpenCl parallel programming paradigm. This paradigm give us the unified programming model to accelerate algorithms in heterogeneous systems.

We can split the spectral unmixing in three stages. The first has the task of finding the number of pure pixels, called endmembers, pixels formed of a single spectral signature. Will be used the algorithm known as *Geometry-based Estimation of number of endmember*, **GENE**. The second stage is responsible for identifying endmember pixel and extract them along with all its bands. For this purpose we choose the algorithm called *Simplex Growing Algorithm*, **SGA**. In the last stage, making abundance maps of each endmember will be created. This step will be charged the algorithm known as, *Sum-to-one Constrained Linear Spectral Unmixing*, **SCLSU**.

Three platforms have been used in this paper: As **CPU**, Intel Xeon E5-2695 v3, **GPU**, NVidia GeForce GTX 980 and as **Accelerator**, Intel Xeon Phi 31S1P. The main idea is compare each platform and evaluate which one fits best our needs.

Keywords

Hyperspectral imaging, spectral unmixing, parallel computing, OpenCl, heterogenous systems.

Capítulo 1

Introducción

1.1. Motivación

En la época actual, los dispositivos tecnológicos se encuentran evolucionando continuamente, lo que obliga al software implementado para ellos, a avanzar de manera progresiva también. De este modo se requiere optimizar el rendimiento en base a las nuevas características que proporciona el hardware.

En este proyecto tratamos justo este tema, el avance de la tecnología, enfocado más concretamente al tratamiento eficiente de una serie de muestras digitales. Estas muestras son recogidas por un tipo de sensores específicos, llamadas imágenes hiperespectrales [1, 2]. Estas imágenes, se consideran una ampliación de las imágenes digitales, puesto que cada píxel tiene un amplio conjunto de valores, los cuales corresponden a las diferentes mediciones espectrales, siendo este un valor discreto en las imágenes digitales.

Dichas imágenes son tomadas de manera remota por sensores situados en satélites o en plataformas aéreas, cuya finalidad es poder representar áreas de la superficie terrestre. Estas imágenes se basan en la medición de la radiación reflejada por los diferentes materiales de la superficie terrestre, cada uno con una longitud de onda especifica y una firma espectral definida.

El análisis de imágenes hiperespectrales es empleado para multitud de cometidos, entre ellos comprobar la contaminación tanto en aire como agua de un entorno geográfico. Moni-

torizar incendios, en este apartado es de vital importancia el procesamiento de las imágenes en tiempo real, de esta forma se pueden controlar de manera más eficaz. Identificar las diferentes especies de un ecosistema, con el fin de contabilizarlas y poder realizar estudios. Analizar la pureza de los materiales de una zona, de esta manera se pueden realizar prospecciones más precisas. No podemos olvidarnos de sus aplicaciones militares, como situar objetivos o detectar minas en zonas de guerra.

Una de las técnicas más utilizadas para llevar a cabo el análisis de la composición y obtención de las diferentes firmas espectrales que se encuentran en ella es el desmezclado espectral. Con esta técnica se permite determinar si la composición de un píxel es un elemento puro (designado con el nombre de *endmember* [3]) o si se encuentra formado por un conjunto de elementos puros o endmembers.

El problema de esta técnica es el tiempo que conlleva realizarla, esto se debe al gran tamaño que ocupan las imágenes, lo que también nos lleva a tener problemas de almacenamiento. Para resolver estos imprevistos, inicialmente se hicieron uso de clusters, aunque actualmente no parece la solución definitiva.

Nosotros proponemos otra solución diferente, basada en los avances de las unidades de procesamiento gráfico (*GPUs*) [4]. Las *GPUs* son unidades que incorporan multitud de pequeños procesadores con la capaz de realizar procesos de manera simultanea. En los últimos años, el mercado de los videojuegos se ha convertido en un fuerte sector, lo que hace que las *GPUs* evolucionen. Del mismo modo, las unidades de procesamiento central (*CPUs*) han evolucionado también, ofreciendo otras prestaciones.

En esto se centra nuestro trabajo, en aprovechar las altas prestaciones desde el punto de vista de rendimiento que aporta este hardware. Esto se debe a su forma de tratamiento de datos, la cual se realiza de manera paralela, lo que nos permite tratar un mayor número de datos en el mismo tiempo. Pudiendo llegar a conseguir el procesamiento de las imágenes en tiempo real.

1.2. Objetivos

El objetivo de este proyecto de fin de grado ha consistido en la aceleración de algoritmos de procesamiento de imágenes hiperespectrales. En el cual tratamos de resolver dos importantes problemas que son el problema de la mezcla espectral[5] y el gran tamaño de los datos. Las soluciones actuales son inviables para un sistema de procesamiento en tiempo real, así como para almacenar la gran cantidad de datos tomados por el sensor. Lo que propone el presente proyecto para abordar dichas dificultades, son la validación e implementación de forma paralela de una serie de algoritmos, obteniendo de esta manera un procesamiento de imágenes hiperespectrales en tiempo real.

En este trabajo de investigación se han estudiado, diseñado e implementado un algoritmo para el cálculo de endmembers (Geometry-based Estimation of number of endmembers -*GENE*), un algoritmo para la extracción de endmembers (Simplex Growing algorithm -*SGA*) y uno para la estimación de abundancias (Sum to one Constained Linear Spectral Umnixing -*SCLSU*). Con lo que formamos una cadena completa de desmezclado espectral. La consecución del objetivo general anteriormente mencionado se ha llevado a cabo abordando una serie de objetivos específicos, los cuales se enumeran a continuación:

- En primer lugar realizar un proceso de documentación acerca del análisis hiperespectral (técnicas de desmezclado, imágenes y arquitecturas utilizadas...), de tal manera que permita comprender los algoritmos seleccionados para llevar a cabo su estudio.
- Estudiar de manera exhaustiva los diferentes algoritmos seleccionados para llevar a cabo la paralelización, realizando el diseño más óptimo posible de la partes paralelizables, para su futura implementación.
- Geometry-based Estimation of number of endmembers (*GENE*):
    - Realizar un estudio exhaustivo del funcionamiento del algoritmo encargado de la estimación del número de endmembers, para conseguir un diseño eficiente de su

implementación en paralelo.

  - Implementar el algoritmo para su futura ejecución en diferentes plataformas, lo cual hace que nos decantemos por el uso del lenguaje de programación paralela OpenCl. Esto se debe a una de sus grandes virtudes (soporta aceleradores, unidades de procesamiento gráfico ($GPUs$) y unidades centrales de procesamiento ($CPUs$)). Al mismo tiempo hacemos uso de librerías, que implementan funciones optimizadas, requeridas en el funcionamiento del algoritmo.
  - Efectuar una comprobación tanto de tiempos como de resultados sobre datos reales y sintéticos. Y de esta manera obtener una comparación con una versión original en serie, y la versión desarrollada en paralelo. Así podemos analizar la mejora obtenida en el rendimiento.

- Simplex Growing algorithm (SGA):

  - Elaborar un estudio sobre el algoritmo de extracción de endmembers (SGA), con el fin de distinguir las partes paralelizables.
  - Implementar la solución en paralelo, basándonos en el diseño hecho previamente. Puesto que el fin del proyecto es llevar a cabo un estudio de los resultados obtenidos en diferentes plataformas, hacemos uso del lenguaje OpenCl para su desarrollo.
  - Realizar un estudio con diferentes ejecuciones sobre datos producidos a partir de imágenes reales y sintéticas, y compararlos con la versión original en serie. De esta manera, podemos concluir el rendimiento de la paralelización del algoritmo, así como la precisión del mismo.

- Sum to one Constained Linear Spectral Umnixing ($SCLSU$):

  - Realizar un estudio del algoritmo de estimación de abundancias ($SCLSU$), en el cual planteamos las optimizaciones posibles a realizar en la paralelización.

    - Llevar a cabo la implementación de la optimización en paralelo. Para este caso usamos librerías, basadas en opencl, más en concreto clMAGMA y ViennaCL. Esto se debe a que las funciones paralelizables se encuentran implementadas en dichas librerías, y sería difícil obtener un mejor rendimiento.
    - Cotejar mediante una serie de ejecuciones con datos proporcionados por imágenes sintéticas y reales, los resultados obtenidos tanto en precisión como en rendimiento de nuestro algoritmo paralelizado respecto al original.

- Con todos los algoritmos paralelizados y realizada la comprobación de que su rendimiento y precisión son óptimos. Aunar los algoritmos, obteniendo una cadena completa de desmezclado espectral.

- Ejecutar nuestra cadena de desmezclado espectral en diferentes plataformas, en concreto emplearemos una *CPU Xeon E5-2695 v3*, una *GPU NVidia GTX 980* y un *Acelerador Xeon Phi 31S1P*. Con ello queremos obtener una comparación del rendimiento adquirido en cada plataforma, lo que nos permitirá identificar que plataforma nos proporciona un mejor rendimiento.

- Obtener conclusiones a partir del estudio cuantitativo y comparativo realizado, y plantear posibles trabajos futuros.

Capítulo 2

Análisis hiperespectral

2.1. Imágenes hiperespectrales

Las imágenes hiperespectrales reciben su nombre por la cantidad de canales espectrales que las componen. Se dividen en bandas, cada banda se corresponde a un espectro. Con longitudes de onda variables entre ultravioleta (unos 400 nm) e infrarrojo (unos 2500 nm). Las bandas suelen estar distribuidas a lo largo de todo el espectro medible por el sensor, esto es lo que las diferencia de las imágenes multiespectrales, las cuales almacenan parte de los espectros medibles [6]. Además las imágenes multiespectrales[7] suelen captar longitudes de ondas correspondientes a ciertos espectros concretos. Siguiendo con la comparación las imágenes normales solamente están compuestas por tres canales espectrales: rojo, con una longitud de onda entre los 625 y los 740nm; verde, con una longitud de onda entre los 520 y los 570 nm; y azul, con una longitud de onda entre los 440 y los 490 nm.

La resolución espectral tan alta frente a los otros tipos de imágenes, da como resultado que los píxeles tengan multitud de componentes diferentes (recordemos que en las imágenes normales se componen de tres componentes). Es esto lo que lleva a denotar un vector de datos cuando se habla de un píxel en una imagen hiperespectral, este vector es conocido como firma espectral.[8]

El resultado de los datos recogidos por el sensor hiperespectral en una escena determinada puede interpretarse como un cubo de datos. De este cubo se utilizan dos dimensiones para

la localización de los diferentes píxeles en la escena (siendo estas generalmente líneas y muestras), mientras que la tercera dimensión se corresponde con la particularidad espectral en diferentes longitudes de onda de cada píxel.[9]

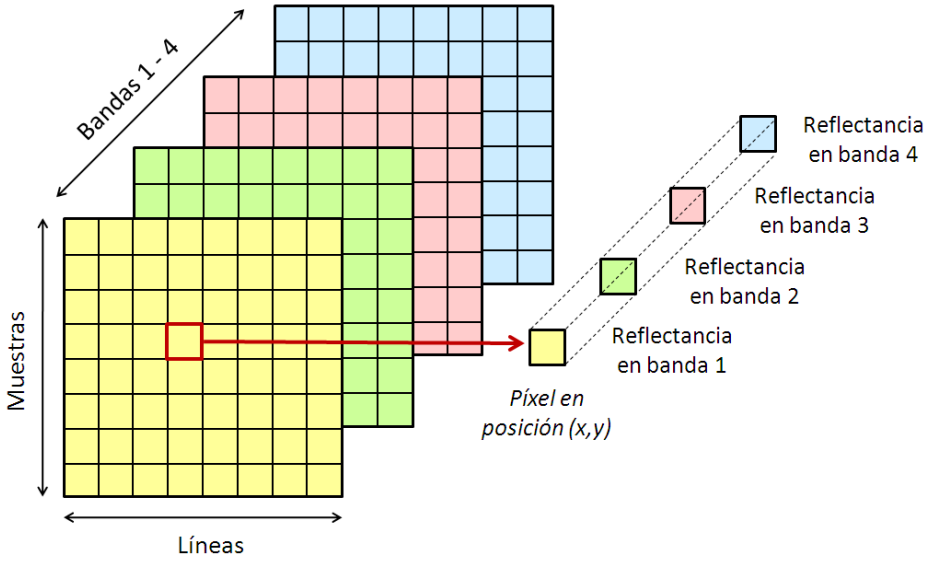

Figura 2.1: *Esqueleto de una imagen hiperespectral.*

En la figura (2.1) podemos visualizar la estructura comentada, donde el eje X se corresponde con las líneas, el eje Y indica las muestras. Por último, el eje Z es el que muestra las bandas (la longitud de onda en ese canal espectral).

La firma espectral de cada píxel está caracterizada por su forma de reflectar la luz en las distintas longitudes de onda en las que esta tomada la muestra. De esta forma podemos caracterizar los materiales en una imagen. Puesto que cada uno reflectará de un modo particular los rayos de luz en función de la longitud de onda.

En la figura (2.2)[1], se puede ver un ejemplo de un sensor colocado en un satélite, desde el cual se toma una muestra de la Tierra, obteniendo la firma espectral de un píxel de la imagen tomada. En dicha figura también se aprecia la diferencia entre las imágenes hiperespectrales y las imágenes multiespectrales, comentadas anteriormente.

[1]http://opticalnanofilter.com/

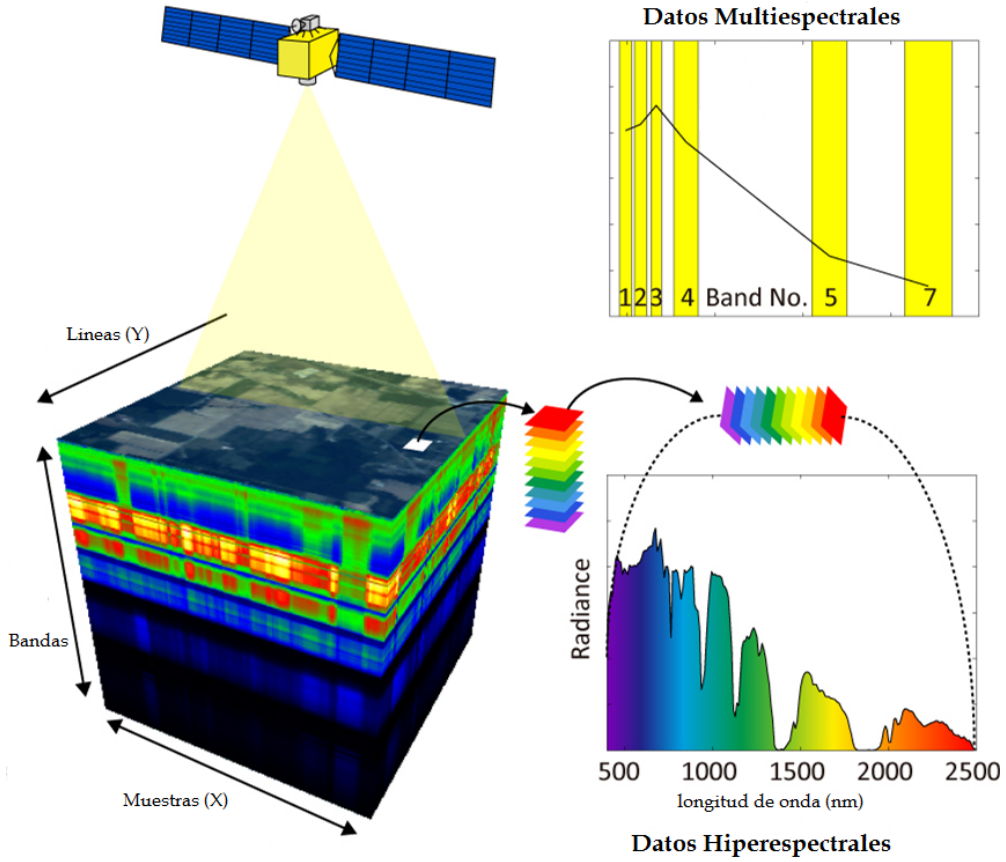

Figura 2.2: *Ejemplo de una muestra tomada con un satélite.*

Como comentamos en la introducción, actualmente los sensores para la toma de imágenes hiperespectrales, conocidos como espectómetros, se suelen situar en aviones no tripulados o satélites.

En nuestro caso usamos imágenes hiperespectrales tomadas con el sensor AVIRIS(Cuprite y SubsetWTC). *AVIRIS*[2] es el acrónimo de Airborne Visible InfraRed Imaging Spectrometer (Espectrómetro de imágenes visibles-infrarrojas aerotransportado). Se trata de un sensor hiperespectral situado en diferentes plataformas capaz de analizar zonas visibles e infrarrojas del espectro[10, 11]. Este sensor se encuentra operativo desde el año 1987, con capacidad de obtener información en 224 canales espectrales contiguos, con un ancho entre las bandas de 10 nm, cubriendo un rango de longitudes de onda entre 400 y 2500 nm.

[2]http://aviris.jpl.nasa.gov

El objetivo principal del proyecto AVIRIS es identificar, medir y controlar los componentes de la superficie y la atmósfera de la Tierra sobre la base de firmas de absorción y dispersión de partículas moleculares. La investigación con datos de AVIRIS se centra en los procesos relacionados con la comprensión del medio ambiente global y el cambio climático.

El sensor ha recogido muestras de datos en Canadá, Estados Unidos y Europa. Para llevar a cabo dicha recogida de datos, el sensor se instaló en dos aeronaves:

- El avión ER-2[3] perteneciente al Jet Propulsion Laboratory de la NASA con una velocidad máxima de 730 km/h, el cual puede llegar a una altura de 20 km sobre la superficie terrestre.
- El avión Twin Otter[4] desarrollado por la compañía canadiense Havilland Canadá con una velocidad máxima de 130 km/h, alcanzando una altura de 4km de altura sobre el nivel del mar.

2.2. Problema principal: mezcla espectral

En las imágenes hiperespectrales es inevitable encontrar una mezcla de los distintos materiales que se hallan en la amplitud de la muestra independientemente de la resolución espacial de la misma. Por ello existen dos tipos de píxeles: puros y mezclas[12].

Siendo los píxel mezcla[13] aquellos en los que encuentran diversos materiales diferenciados entre sí. Los píxel mezcla son los más cuantiosos en las imágenes hiperespectrales, ya que es un fenómeno que se da a niveles microscópicos.

En la figura (2.3)[5] se visualiza un ejemplo de la toma de píxeles en una muestra hiperespectral, donde podemos contemplar píxeles puros a la par que píxeles mezcla.

Un ejemplo de mezcla es el píxel comprendido por las rocas, en el cual podemos encontrar rocas y suelo. En este caso, el espectro medido se puede descomponer en una combinación

[3]https://airbornescience.nasa.gov/aircraft/ER-2
[4]https://airbornescience.nasa.gov/aircraft/Twin_Otter_-_GRC
[5]http://www.umbc.edu/rssipl/people/aplaza/PFC_Victor_Fermin.pdf

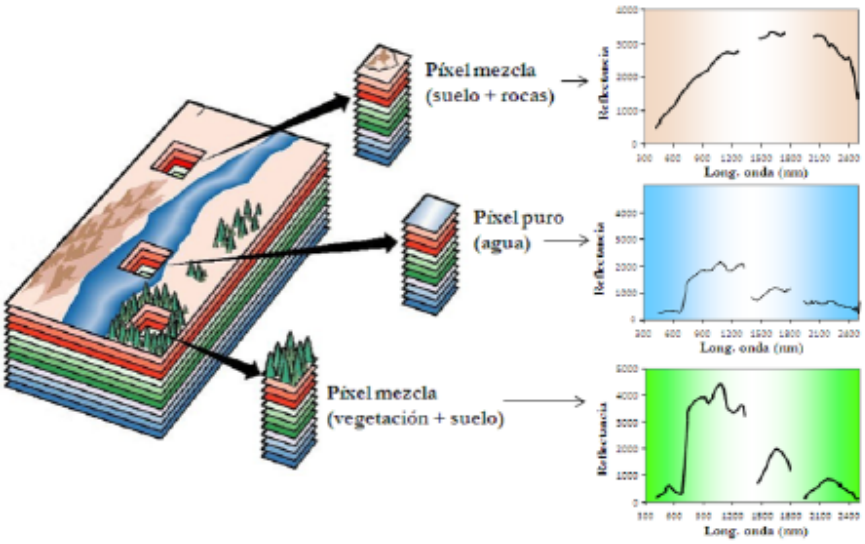

Figura 2.3: *Muestra con diferentes tipos de píxel.*

de firmas espectrales diferenciando la porción de cada firma macroscópica pura en el píxel mezcla [14]. Por otro lado está el caso del agua, este tipo de materiales se consideran puros, aunque pueden variar su firma espectral por diferentes condicionantes (ángulo de observación, iluminación)[15].

Cuando se analiza y se trata una imagen hiperespectral, lo que se desea es determinar los distintos materiales que la componen, así como la cantidad de los mismos. En los píxeles puros es una tarea mas simple, pues tienen la firma espectral del material que se encuentre en dicho píxel, esto varía en los píxeles mezcla, donde es una combinación de los que componen ese píxel. Este fenómeno es lo que se conoce como el problema de la mezcla espectral. Para poder solucionar este problema se lleva a cabo un tratamiento de las imágenes, conocido como desmezclado espectral[14]. Este tratamiento consiste en una serie de estrategias con el fin de identificar el número de endmembers en la imagen, sus respectivas firmas espectrales[16] y la abundancia de estas firmas en los diferentes píxeles de la escena[17]. Estas estrategias se basan en un sistema de ecuaciones que conforman una cadena de desmezclado[18], permitiendo extraer la información de imágenes sin supervisión

a nivel de subpíxel[19].

Antes hemos mencionado los términos endmember y abundancias, se define el termino endmember como la representación de materiales espectralmente puros presentes en la imagen, mientras que las abundancias representan el reparto de los endmembers por la imagen [14] [18], la proporción de cada uno de los endmembers en cada píxel de la imagen hiperespectral.

Según la imagen hiperespectral a analizar y el caso concreto, puede que se deba cambiar un poco la definición de ambas cosas.

En el caso de los endmembers, aunque un material sea puro, quizás no interese definirlo como endmember, esto es algo que depende del problema en cuestión. Con un ejemplo se entiende mejor, supongamos una imagen hiperespectral de una pradera con tres tipos de setas diferentes, dos tipos de piedras diferentes y hierba. De aquí se puede deducir que tenemos seis endmembers. Esto varia si lo que queremos es analizar las zonas donde haya más setas, con lo que nos bastará con tres endmembers, uno para indicar las setas, otro para indicar las piedras y el restante para definir la hierba. Con lo que podemos concluir, que dependiendo el estudio a realizar se puede considerar una definición de endmember concreta, de tal modo que se ajuste lo más posible a las necesidades de la investigación.

Por otro lado, la abundancia de un endmember en una imagen hiperespectral está sujeta a ciertos factores. La abundancia de un endmember en una imagen hiperespectral es el porcentaje que hay de ese endmember en dicha imagen. El cálculo de la misma se basa en una función lineal, en la cual no se tiene en cuenta que los materiales no reflejan del mismo modo la luz. Con lo que si hay un material que no refleja la luz, pero predomina en un píxel con respecto a otro que si la refleja bien, se realizará una estimación errónea de la cantidad de ambos materiales en ese píxel. A pesar de que la estimación del porcentaje de radiación sea correcto. Otros factores a tener en cuenta serían la humedad en el ambiente, dado que los materiales mojados no reflectan de la misma manera la luz. Las condiciones climáticas son otro agente influyente, puesto que una gran cantidad de nubes, pueden imposibilitar que

los rayos de luz alcancen de manera adecuada el suelo. No obstante, la definición tradicional es válida como demuestran diferentes cadenas de desmezclado.

Los algoritmos de desmezclado se basan en los modelos de mezcla, que se pueden dividir en modelos lineales y no lineales[20]. Los modelos lineales de mezcla asumen que los reflejos secundarios son despreciables, así como efectos de dispersión. Esto da como resultado una combinación lineal de las firmas espectrales puras de los materiales en los píxeles mezcla [21]. Los modelos no lineales[22] pueden caracterizar mejor el espectro de mezcla resultante en determinados casos ya que tienen presentes las interacciones físicas en la luz reflejada por los distintos materiales de la muestra.

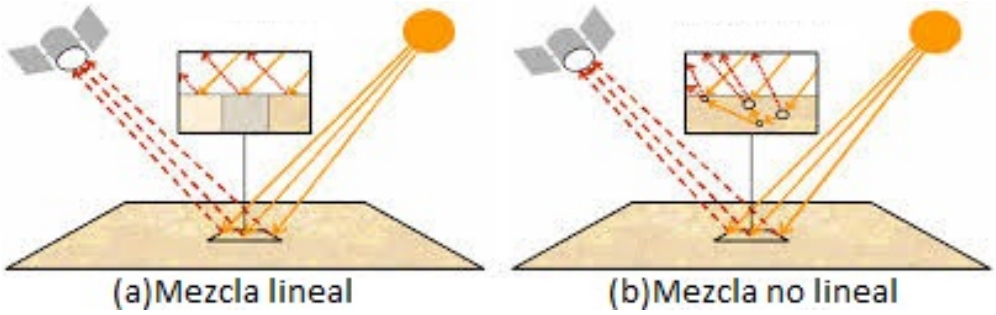

Figura 2.4: *Ejemplo de los modelos de mezclado.*

En la figura (2.3)[6] se pueden apreciar los diferentes modelos, la figura (2.3 a) es el modelo lineal, en él se visualiza como los rayos solares son reflejados según llegan a la superficie del material. Por otro lado, en el modelo no lineal, la figura (2.3 b), se aprecia como los rayos se dispersan primero sobre la superficie de los materiales antes de ser reflejados. Otro caso sería el de tener vegetación, y que al reflejar la luz del suelo, topara con esta vegetación, alterando de esta forma la radiación que el sensor captará y con ello los resultados de dicha muestra.

[6]://www.umbc.edu/rssipl/people/aplaza/Papers/Conferences/2011.SPIE.Parallelunmixing.pdf

2.3. Necesidad de paralelismo en imágenes hiperespectrales

Como comentamos en la introducción, las imágenes hiperespectrales tienen un gran problema en lo que al tamaño se refiere. Dicho problema afecta al tratamiento y almacenamiento de las mismas, la falta de arquitecturas consolidadas para el tratamiento eficiente de las imágenes hace que instituciones como NASA, que recopila varios Terabytes de información al día, guarde un gran porcentaje de estos datos con la idea de que en un futuro puedan ser procesados. Estos datos en muchas ocasiones no llegan a ser procesados debido a los grandes requerimientos computacionales que conlleva almacenarlos, procesarlos y distribuirlos de manera eficiente.

Para solventar este problema hay diferentes opciones. Entre ellas se pueden encontrar algunas radicales con el fin de reducir el tamaño de los datos tomados por el sensor. Un ejemplo de estas es el caso de recopilar la información por demanda de los usuarios, lo cual limita el número de bandas a las necesarias para poder llevar a cabo el estudio requerido. Del mismo se limita también el número de muestras y líneas, dado que se puede concretar la zona específica. Con esta solución se pierde información, obteniéndose la realmente necesaria para el estudio demandado, de tal manera que la imagen tendrá un menor tamaño, y será más fácil de enviar desde el sensor, procesar, almacenar y distribuirla.

Otra opción sería el uso de computación cluster[23], este método consiste en recopilar las muestras por los sensores remotos de observación hiperespectral, para su futuro tratamiento en el cluster[24, 25]. No es mala solución para abordar el problema del procesamiento de datos, así como el almacenamiento, pues suelen contar con gran capacidad. Aun así, aun quedaría un problema, y es el de enviar los datos del sensor, puesto que, aunque las redes actuales cuenten con una gran velocidad de transmisión, los datos recogidos ocupan varios Terabytes.

Hay más soluciones, entre ellas la que se propone en este proyecto. La cual consiste en

llevar a cabo el procesamiento de los datos recogidos por el sensor remoto de observación mediante el uso de lenguajes paralelos. Evitando de esta manera la necesidad de enviar los datos a un cluster para tratarlos allí. Esto se puede llevar a cabo gracias a las grandes mejoras en las arquitecturas de computación paralela, lo que permite la realización de un procesamiento eficiente de los datos hiperespectrales. Teniendo en cuenta que los accesos a datos en los algoritmos de análisis hiperespectral son muy predecibles, se puede llevar a cabo un proceso de paralelización parcial de estos algoritmos, paralelizando las partes de mayor carga computacional. Gracias a esto se consiguen grandes mejoras en el rendimiento, sin poner en riesgo la efectividad de dichos algoritmos.

2.4. Cadena de desmezclado propuesta para paralelización

En este proyecto hemos llevado a cabo la paralelización de una cadena de desmezclado completa, esto consiste en tres algoritmos para completar la estimación del número de endmembers, la extracción de los endmembers y la abundancia de los mismos en la imagen. Para ello hemos seleccionado el algoritmo *GENE* para la estimación del número de endmembers, no es el más empleado, con lo cual supone un reto a la par que ofrece una alternativa a los típicamente usados. Para la extracción de los endmembers elegimos el algoritmo *SGA*, finalmente para la estimación de las abundancias nos decantamos por el algoritmo *SCLSU*.

En la figura (2.5)[7] tenemos el proceso completo de desmezclado de una imagen hiperespectral. Lo primero que obtiene el sensor es una muestra de gran tamaño, con la ayuda de un algoritmo de estimación de número de endmembers se establecen los que hay en la muestra. Tras este paso se puede realizar una reducción de la imagen, esto es opcional, aunque facilita la tarea para el siguiente paso. Con el número de endmembers identificado, toca el turno al algoritmo de extracción de endmembers, encargado de identificar las firmas espectrales de los endmembers. Una vez determinadas las firmas espectrales, se procede a a extraer las

[7] Tesis de Sergio Sánchez Martínez

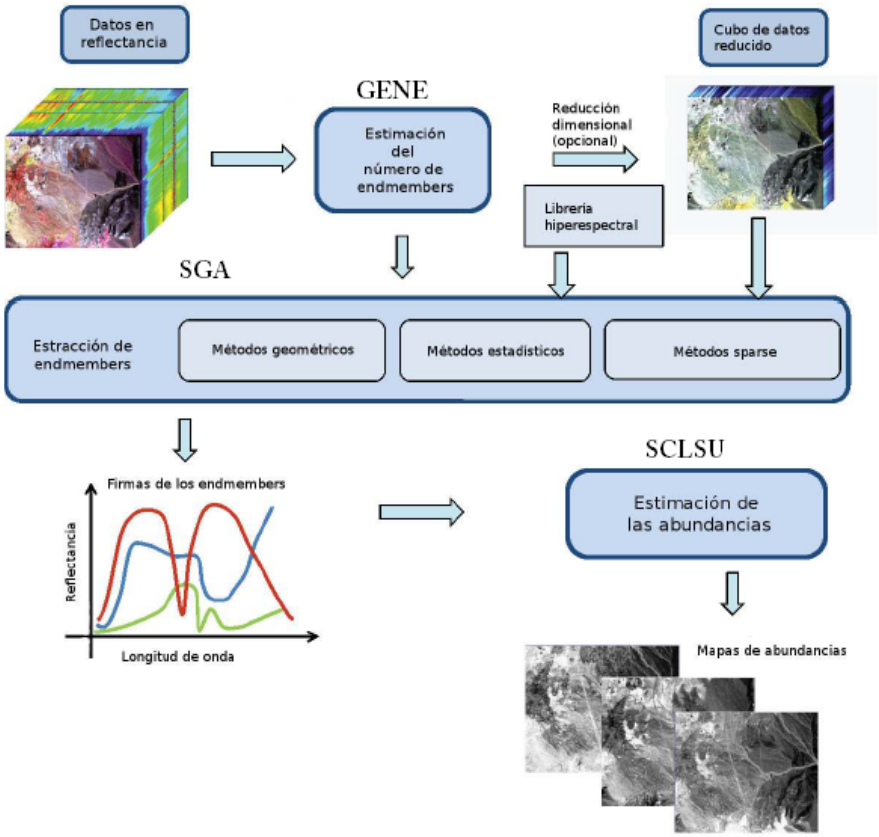

Figura 2.5: *Proceso de desmezclado de una imagen hiperespectral.*

abundancias de los diferentes endmembers encontrados en la imagen.

En este punto vamos a analizar un poco más detalladamente en qué consisten estos tres algoritmos.

2.4.1. Geometry-based Estimation of Number of Endmembers (GENE)

El método GENE hace uso de las características geométricas clave de los datos hiperespectrales. La propuesta original introdujo dos métodos diferentes denotados como GENE-CH y GEN-AH, respectivamente. La primera supone la presencia de píxeles puros en los datos mientras que la segunda no hace esta suposición.En este trabajo nos hemos centrado en GENE-AH (ver algoritmo 1), ya que es más robusto que el GEN-CH contra la ausencia de

píxeles puros en los datos.

Algorithm 1 Pseudocódigo para el algoritmo GENE

Require: $\mathbf{X}, \mathbf{W}, \mathbf{N} > \mathbf{3}, \mathbf{P\_FA} \geq \mathbf{0}$
Ensure: $\hat{M}, \hat{p}$

$\mathbf{C_w} := \frac{(\mathbf{W}-\bar{\mathbf{W}})(\mathbf{W}-\bar{\mathbf{W}})^\mathbf{T}}{\mathbf{n_s}}$
$\mathbf{C_x} := \frac{(\mathbf{X}-\bar{\mathbf{X}})(\mathbf{X}-\bar{\mathbf{X}})^\mathbf{T}}{\mathbf{n_s}} - \mathbf{C_w}$
$\mathbf{U} := \mathbf{U} \sum \mathbf{U^T} \equiv \mathbf{C_x}$
$\tilde{\mathbf{X}} := \mathbf{U}(\mathbf{X} - \bar{\mathbf{X}})$
$\tilde{\mathbf{C}}_\mathbf{w} := \mathbf{U}\mathbf{C_w}\mathbf{U^T}$
$\mathbf{k} := \mathbf{1}$
$\hat{\mathbf{M}}_\mathbf{k} := \tilde{\mathbf{X}}_{\mathbf{l_k}}$ para $l_k \in max_i \ \|\tilde{X}_i\|_2$
$\mathbf{Q} := \tilde{\mathbf{M}}_\mathbf{k}$
for $\mathbf{k} = \mathbf{2}$ **hasta** $\mathbf{k} <$**n do**
  $\tilde{\mathbf{M}}_\mathbf{k} := \tilde{\mathbf{X}}_{\mathbf{l_k}}$ **para** $l_k \in max_i \ \|\tilde{P}_Q \perp X_i\|_2$
  $\theta := \mathbf{argmin}_{\mathbf{1}^\mathbf{T}_{\mathbf{k-1}}\theta=\mathbf{1}} \|\hat{\mathbf{M}}_\mathbf{k} - \theta\|^\mathbf{2}_\mathbf{2}$
  $\mathbf{e} := \hat{\mathbf{M}}_\mathbf{k} - \mathbf{Q}\theta$
  $\mathbf{r} := \mathbf{e^T}((\mathbf{1} + \theta^\mathbf{T}\theta)\tilde{\mathbf{C}}_\mathbf{w})^{-\mathbf{1}}\mathbf{e}$
  $\psi := \mathbf{1} - \frac{\gamma(\mathbf{r/2},(\mathbf{N}-\mathbf{1})/\mathbf{2})}{\mathbf{\Gamma}((\mathbf{N}-\mathbf{1})/\mathbf{2})}$
  if $\psi > P\_FA$ **then**
    $\tilde{\mathbf{M}} := \mathbf{Q}$
    $\mathbf{k} := \mathbf{1}$
  end if
  $\mathbf{Q} := [\mathbf{Q}, \hat{\mathbf{M}}_\mathbf{k}]$
end for

Este algoritmo recibe como parámetros de entrada X, W, N, P_FA. Siendo X una matriz con los datos hiperespectrales a procesar, de tamaño Lxn_s la L se corresponde con el número de bandas, mientras que n_s se corresponde al número de píxeles. W se corresponde con la matriz de la estimación del ruido de la escena, es de tamaño $L\,x\,n_s$ también. Los otros dos parámetros de entrada N, P_FA pertenecen a el número máximo de endmembers de la escena y la probabilidad de fallo respectivamente.

En este trabajo se ha utilizado el método de estimación de covarianza de ruido basado en el análisis de la multiplicación de regresión reportado en $lineas1y2$ como lo sugieren los autores del GENE. La reducción de dimensionalidad es similar al cálculo de los principa-

les componentes de los datos, pero con la eliminación de las estadísticas de ruido de los datos de la matriz de covarianza. El algoritmo *GENE* puede ser utilizado en combinación con cualquier método de extracción endmember, pero los autores proponen un algoritmo de extracción de endmember basado geometría llamado *TRI-P*. Como ATGP[26], TRI-P identifica endmembers iterativamente: en cada iteración el algoritmo identifica un nuevo endmember potencial, que GENE intenta descomponerse en una combinación lineal de los endmembers previamente identificados. Al utilizar el error residual de la descomposición y de las estadísticas de ruido de los datos, el algoritmo lleva a cabo una regla de clasificación Neyman-Pearson para determinar si la dimensionalidad del nueva subespacio es más alta que la de la iteración anterior, es decir, la salida del clasificador Neyman-Pearson se utiliza como criterio de interrupción GENE.

2.4.2. Simplex Growing Algorithm (SGA)

El algoritmo original SGA fue desarrollado en [27] para propósitos de desmezclado espectral. Era una alternativa al algoritmo N-finder [28] y demostró ser una prometedora técnica de extracción de endmembers. Este algoritmo pertenece a la segunda etapa del proceso de desmezclado, la etapa de extracción de endmembers (EEA)[8].

El algoritmo (2) recibe como parámetros de entrada una matriz Y y un entero $\hat{p}$. Esta matriz Y se corresponde con los datos hiperespectrales a procesar, de tamaño $L\ x\ n_s$ la L se corresponde con el número de bandas, mientras que n_s se corresponde al número de endmembers. El parámetro $\hat{p}$ se corresponde con el número estimado de endmembers que tiene la muestra. Esta estimación se consigue con los algoritmos de la primera etapa del proceso de desmezclado, los de estimación del número de endmembers (en nuestra cadena, el algoritmo SCLSU). El algoritmo asume la presencia de estos endmembers en la imagen, donde inicialmente genera un endmember desde un píxel t aleatorio. Los resultados experimentales demuestran que no influye sobre el resultado final el píxel t aleatorio seleccionado.

[8]Endmember Extraction Algorithms

Los siguientes endmembers son producidos por la generación de volumen definido en el paso 4, donde se selecciona el máximo de todos ellos. Esta generación de volumen se repite hasta obtener el número deseado de endmembers p. Finalmente obtenemos un conjunto de endmembers, siendo esto lo que devuelve el algoritmo.

Algorithm 2 Pseudocódigo para el algoritmo SGA

Require: Y, $\hat{p} > 0$
Ensure: $\hat{E}$

1: $n = 1$
2: $\mathbf{e_n} := \mathbf{arg}\left\{ \underset{\mathbf{r}}{\mathbf{Max}} \left[\left| \mathbf{det} \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{t} & \mathbf{r} \end{bmatrix} \right| \right] \right\}$
3: $\hat{E}_n := [e_n]$
4: **for** n hasta $n < p - 1$ **do**
5: $\mathbf{V(e_1, \cdots, e_n, r)} := \dfrac{\left| \mathbf{det} \begin{bmatrix} \mathbf{1} & \mathbf{1} & \cdots & \mathbf{1} & \mathbf{1} \\ \mathbf{e_1} & \mathbf{e_2} & \cdots & \mathbf{e_n} & \mathbf{r} \end{bmatrix} \right|}{\mathbf{n!}}$
6: $\mathbf{e_{n+1}} := \mathbf{arg}\left\{ \underset{\mathbf{r}}{\mathbf{Max}} \left[\mathbf{V(e_1, \cdots, e_n, r)} \right] \right\}$
7: $\mathbf{\hat{E} := [\hat{E}, e_{n+1}]}$
8: **end for**

2.4.3. Sum-to-one Constrained Linear Spectral Unmixing (SCLSU)

Por ultimo el algoritmo *SCLSU*, (algoritmo 3), se encarga de generar el mapa de abundancias de cada *endmember* extraído por el algoritmo anterior.

Como parámetros de entrada tenemos la imagen hiperespectral original, Y, junto con los endmembers extraídos en el paso anterior, $\hat{M}$. Como parámetros de salida tenemos la matriz de abundancias. Se generan tantas bandas de abundancias como endmembers encontrados, cada banda corresponde con uno diferente.

El algoritmo SCLSU impone la ASC (Abundance sum-to-one constraint, restricción en las abundancias de suma a uno), sin tener en cuenta el ANC (Abundance nonnegativity constraint, restricción en las abundancias de no negatividad). Como resultado, este método generalmente no estima fracciones de abundancia de endmembers con precisión. Sin embar-

go, sus fracciones de abundancia estimadas se pueden utilizar para el propósito de detección de endmembers. A las fracciones de abundancia generadas por SCLSU se les ha de sumar uno, cuando una escena de la imagen contiene muchas firmas de distintos endmembers, como es el caso de las imágenes hiperespectrales, donde las magnitudes de las fracciones de abundancia de los endmembers detectados se extienden a lo largo y ancho de la imagen.

Algorithm 3 Pseudocódigo para el algoritmo SCLSU

Require: Y, $\hat{M}$, **lines, samples, bands**
Ensure: **mapas de abundancia**

1: $\mathbf{\bar{X}} = \sqrt[2]{\sum_{i=0}^{lines*samples*sbands} image_i^2}/(\mathbf{lines} * \mathbf{samples} * \mathbf{bands})$
2: $\forall \mathbf{i} \in \mathbf{Y_i} := \mathbf{Y_i}/\mathbf{\bar{X}}$
3: $\forall \mathbf{i} \in \mathbf{\hat{M}_i} := \mathbf{\hat{M}_i}/\mathbf{\bar{X}}$
4: $\mathbf{MtM} := \mathbf{M'} * \mathbf{M}$
5: $\mathbf{MtM} := \mathbf{MtM} \sum \mathbf{MtM} \equiv \mathbf{UF}, \mathbf{SF}$ **(SVD decomposition)**
6: $\mathbf{IFS} := \mathbf{UF}/(\mathbf{SF} + \mu)$
7: $\mathbf{IF} := \mathbf{IFS} * \mathbf{UF'}$
8: $\forall \mathbf{i} \in \mathbf{0} \leq \mathbf{i} < \mathbf{Nmax}, \mathbf{sumaFilas_i} := \sum_{\mathbf{j=0}}^{\mathbf{Nmax}} \mathbf{IF_j}$
9: $\mathbf{sumTotal} := \sum_{\mathbf{i=0}}^{\mathbf{Nmax}} \mathbf{sumFilas_i}$
10: $\forall \mathbf{i}, \mathbf{j} \in \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{Nmax}, \mathbf{IF1_{i*Nmax+j}} := \mathbf{IF_{i*Nmax+j}} - \frac{\mathbf{sumFilas_i} * \mathbf{sumFilas_j}}{\mathbf{sumTotal})}$
11: $\mathbf{Aux_i} := \frac{\mathbf{SumFilas_i}}{\mathbf{SumTotal}}$
12: $\mathbf{YY} := \mathbf{Y} * \mathbf{\hat{M}}$
13: $\mathbf{X} := \mathbf{YY} * \mathbf{IF1}$
14: $\forall \mathbf{i} \in \mathbf{0} \leq \mathbf{i} < \mathbf{lines} * \mathbf{samples}, \mathbf{abundancias_i} := \mathbf{X_i} + \mathbf{Aux_i}$

Capítulo 3

Implementación

3.1. Paradigma de programación paralela OpenCL

Para poder llevar a cabo nuestro desarrollo sobre *OpenCl* primero conviene aclarar algunos conceptos que usaremos más adelante, como son el de *host* y *device*.

El término *host* o anfitrión, se emplea para referirse a un computador cuando éste está conectado a otros dispositivos, teniendo éste datos accesibles por los dispositivos conectados. En este caso el término host es más en concreto el *CPU*.

El término *device* se refiere a los dispositivos sobre los que se programa el código *opencl*, los cuales están conectados de manera física al *host*, o de manera remota por red.

OpenCl es un lenguaje de programación paralela, es multiplataforma, esto significa que se puede portar a distintos dispositivos. Para poder garantizar esta portabilidad del código, posee un modelo de memoria abstracto que se puede implementar en el hardware real.

En *OpenCl*, como en otros lenguajes de programación paralela, la ejecución del programa se produce de manera concurrente, esto significa que no solo hay un flujo de ejecución, como pasa con el código en serie en la *CPU*. Cada uno de estos flujos de ejecución se denomina *thread* o hilo de ejecución, los *thread* se organizan en bloques llamados *workgroup*. Cuando tenemos un código en el *device*, las funciones que se ejecutan allí, se ejecutan por bloques, es decir, se ejecutan de manera simultánea todos los *thread* que pertenecen a ese *workgroup*.

Las funciones que se invocan desde el *host* se denominan *Kernel*, y deben estar declaradas en el *host*, la cabecera de las funciones en el *device* ira precedida del prefijo: $\_\_kernel$.

El tamaño de estos bloques, conocido como local_size, se puede asignar de manera manual, pero teniendo en cuenta:

$$globalmemory = workgroup\_id * local\_size$$

En caso de no asignarlo de manera manual, el compilador le asigna el tamaño al bloque en función de las características del hardware. Para algunos casos concretos es mejor designarlo de forma manual, ya que esto nos permite poder realizar comparaciones y concluir que tamaño tiene mejor rendimiento.

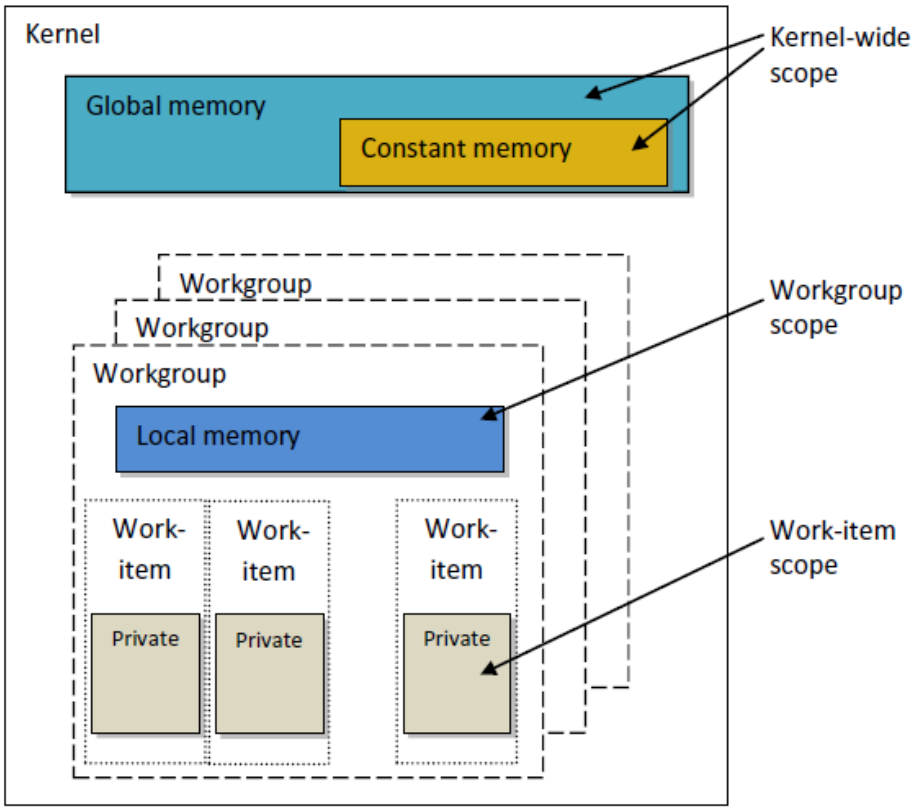

Figura 3.1: *Jerarquía de memoria de memoria en OpenCl, y división en hilos.*

Cuando transmitimos los datos desde la memoria del *host* a la memoria del *device*,

los datos son alojados en la **memoria global** del dispositivo . Esta memoria es accesible por todos los *threads*, independientemente del *workgroup* en el que se encuentran. También podemos tranferirlos a **memoria local**, dicha memoria pertenece a cada bloque. Con esto conseguimos disminuir los accesos a memoria global, lo cual aumenta el rendimiento de nuestro código, pero hay que tener en cuenta que la memoria local es solamente accesible por los *threads* del bloque al que corresponden.

En la figura (3.1)[1] se muestra la distribución en *threads* y en *workgroups* de un *kernel*. Cabe destacar que cada *kernel* tiene su designación concreta del tamaño de los *workgroups* y el número de los mismos. También podemos observar la jerarquía de memoria, como comentábamos antes posee dos tipos: memoria local y la memoria global. Las variables declaradas en el *kernel* son globales, a no ser que se indique lo contrario con el prefijo: *__local*. También se pueden declarar variables locales en la cabecera, reservando la memoria requerida desde el *host*.

3.2. Paralelización de la cadena de desmezclado

En este apartado vamos a desarrollar el proceso de paralelización que llevamos a cabo con los tres algoritmos para obtener la cadena de desmezclado espectral. Debemos explicar de manera exhaustiva cada uno en concreto, dado que su diseño e implementación difieren, como ya vimos en la introducción. Cada algoritmo se ha optado por una forma diferente de optimización, todas ellas basadas en el mismo paradigma de *OpenCl*. Para el algoritmo *GENE* se ha optado por una paralelización basada en el uso de *kernels* escritos en *OpenCl* complementado por la librería *ClMagma*. En el algoritmo *SGA* solo se ha utilizado *kernels* de *OpenCl*. Por último el algoritmo *SCLSU* se ha optado por usar solo librerías, *ViennaCl* y *ClMagma*.

[1] https://www.mql5.com/es/articles/407

3.2.1. Parallel Geometry-based Estimation of Number of Endmembers (P-GENE)

Para este algoritmo, hemos planteado hacer una aceleración híbrida, parte de la aceleración sera proporcionada por la librería *ClMagma* y otra parte por un *Kernel* común creado con *OpenCl*, el problema fundamental que surge es a la hora de crear un espacio, **contexto**, común a ambos métodos.

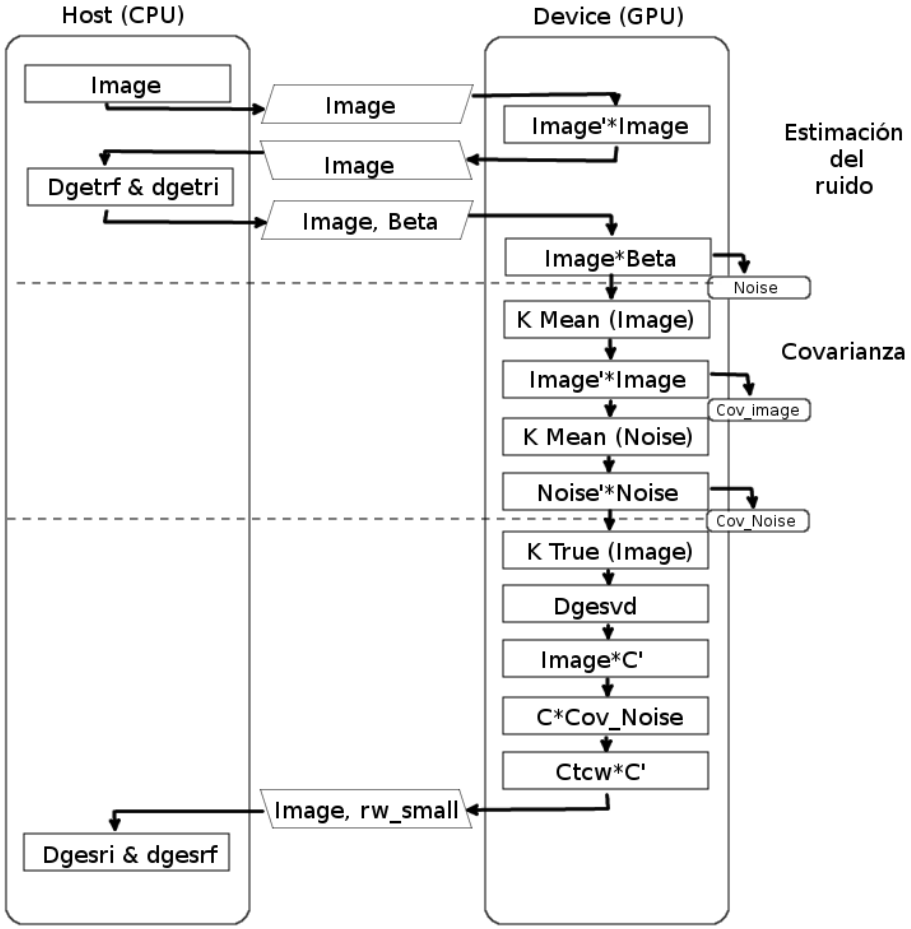

Figura 3.2: *Diagrama de flujo de la primera parte del algoritmo GENE*

Para la explicación se va a dividir el algoritmo en dos partes, la primera parte, figura (3.2), tiene tres subapartados. Como parámetros de entrada tenemos la *imagen hiperespectral*, P_FA, que es la probabilidad de falsa alarma. *Nmax*, número máximo de *endmembers* a

encontrar por el algoritmo. El algoritmo se detiene cuando se supera cualquiera de los dos límites, *P_ FA* ó *Nmax*, a la hora de encontrar los *endmembers*. Una vez termine la ejecución del algoritmo *GENE*, se devolverá el número de *endmembers* que podemos encontrar en la imagen (este será usada por el siguiente algoritmo *SGA*). Además se obtiene una matriz con los *endmembers* encontrados, la cual puede ejecutarse en el algoritmo *SCLSU* y obtener una imagen de abundancias de los *endmembers* extraídos.

Lo primero que necesitamos calcular es la estimación del ruido de la *imagen hiperespectral*. Sabiendo que la imagen es de tamaño *lines*samples*bands*, multiplicamos su transpuesta por sí misma (*magma_ dgemm(image'*image)*). El resultado es una matriz de tamaño *bands*bands*, la imagen resultante la llevamos a *Host*, para calcular su inversa, mediante las funciones **Blas** *dgetri, dgetrf*. Experimentalmente se ha comprobado que calcular la inversa siempre es más rápido en *host* que en *device*. También se calcula la matriz *Beta*, a partir de pequeñas multiplicaciones sobre el resultado de la inversión. Por último, trasladamos las dos matrices al *device*, con el fin de calcular la multiplicación (*magma_ dgemm(Image*Beta)*), obteniendo así la estimación del ruido (*Noise*) en la imagen.

A continuación calculamos la covarianza sobre las matrices *Image* y *Noise*. Para ello es necesario la creación de un *Kernel OpenCl*, ref.(Algoritmo 4). En el proceso sumatorio se utiliza memoria local, para acelerar el resultado, en dos etapas [29].

Algorithm 4 Cálculo del píxel medio por bandas

Require: *bands*, *samples*, *lines*, *Image*
Ensure: La imagen corregida por su píxel medio.

```
for i = 0 hasta i < bands do
    Mean = (sum_{j=0}^{lines*samples} image[i*lines*samples+j]) / (lines*samples)
    for j = 0 hasta j < lines * samples do
        Image[i*lines*samples+j] -= mean
    end for
end for
```

Una vez calculadas las covarianzas de ambas matrices se procede a quitar el ruido (*cov_ noise*) de la imagen (*cov_ image*). Utilizamos un *kernel OpenCl* para el calculo de

(*cov_ image - cov_ noise*). La operación *magma_ dgesvd* sobre la imagen rectificada nos proporciona la descomposición de valores singulares, esta función devuelve tres matrices de tamaño *bands*bands* aunque sólo nos interesa la matriz *C*. Se multiplica *Image*, pero esta vez solo las primeras *Nmax* bandas, por la matriz transpuesta de *C*, (*magma_ dgemm(Image*C')*). Se produce una matriz reducida que llamaremos, *Image_ reduced*, de tamaño *lines*samples*Nmax*. Es entonces cuando multiplicamos la covarianza del ruido con *C* (*magma_ dgemm(c*cov_ noise*c')*), obteniendo así la matriz, *rw_ small*. Por último transferimos las matrices, *Image_ reduced* y *rw_ small* al *Host*, con esta última calculamos su inversa. Tanto *Image_ reduced* como *rw_ small* serán usados en la segunda parte del algoritmo.

La figura (3.3) muestra la segunda parte del algoritmo *GENE*, el propósito es calcular el máximo número de *endmembers* presentes en la *Image*, al mismo tiempo se extraerá una matriz de endmembers.

A partir de la matriz *Image_ reduced* que ya estaba en *Device*, calculamos mediante un *Kernel OpenCl* el píxel más brillante. El cálculo que se lleva a cabo para ello es:

$$\forall \ pxel \ \ (\sum_{i=0}^{bands} img\_reduced_i)^2$$

Le sigue la reducción que se realiza en dos etapas Ref. (Algoritmo 5). La primera transcurre en *Device*, partiendo de un vector tamaño *lines*samples*, el cual se reduce a un vector de tamaño *lines*samples/work_ group* en la primera etapa. Este vector reducido en la primera etapa, es reducido de nuevo en *CPU*. Una vez seleccionado el píxel más brillante (3.4), extraemos sus *Nmax* bandas, creando una matriz a la cual llamaremos *UMatrix*, que sera de tamaño máximo *Nmax*Nmax*. A la matriz se le iran añadiendo bandas de los siguientes endmembers encontrados. Utilizando esta matriz podemos obtener el mapa de abundancias con *SCLSU*.

Hasta aquí hemos calculado el primer *endmember*, es ahora cuando mediante un bucle de *Nmax-1* iteraciones buscaremos el resto de endmembers. El bucle puede detenerse si el *test de Neyman-Pearson* indica que se esta cometiendo un error mayor a *P_ FA*. Dentro del bucle tenemos una etapa compuesta por pequeñas operaciones en serie, estas no deben

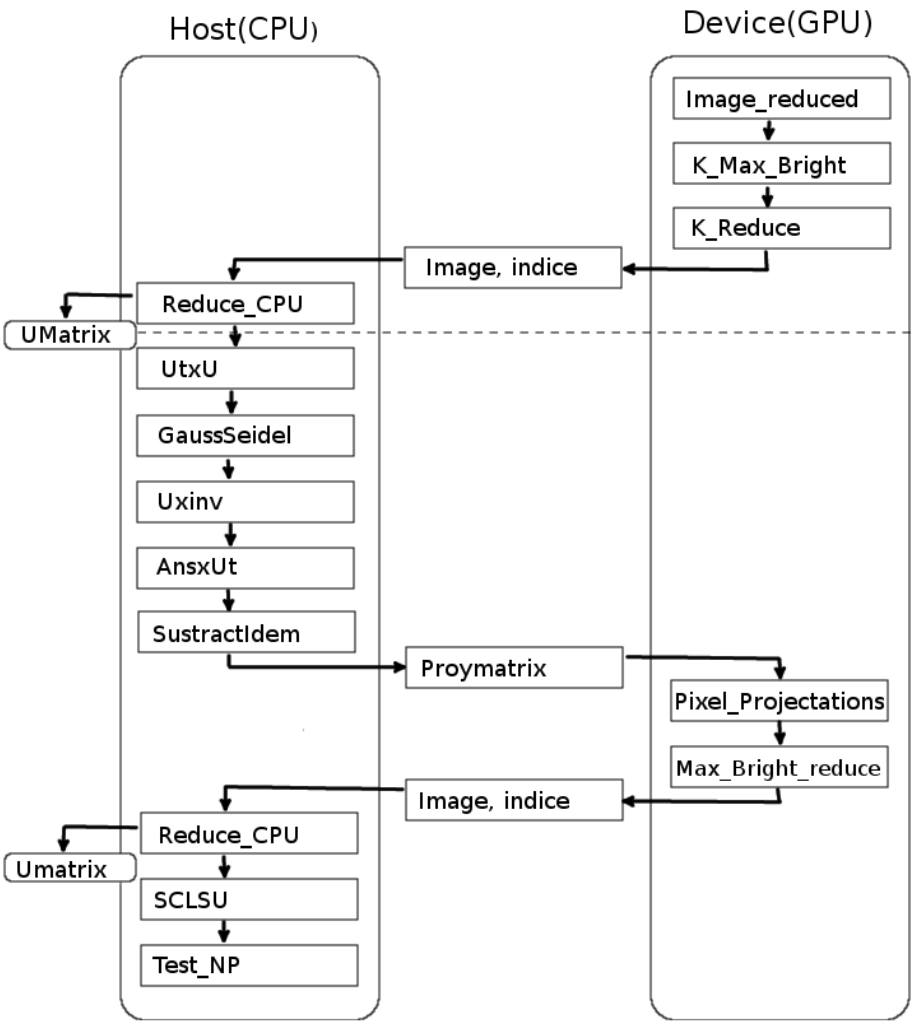

Figura 3.3: *Diagrama de flujo de la segunda parte del algoritmo GENE*

ser objeto de paralelización ya que manejan tamaño *Nmax*Nmax* de matrices. Una vez concluida esta etapa se forma una matriz que mandaremos a *Device* para reducirla en dos pasos (Algoritmo 5). Se han señalado en verde las partes que serán ejecutadas en *GPU*.

Por último, con el píxel encontrado se extrae *Nmax* bandas para rellenar la matriz *Umatrix*. Con el vector del píxel encontrado y la matriz *Umatrix* llamamos a la función *SCLSU*, que será explicada en la sección 3.2.3 para calcular las abundancias, estas son necesarias para calcular el *Test Neyman-Pearson*. Dicho test nos sirve para medir el error que se comete después de añadir el último *endmember* a la matriz *Umatrix*, todo el test es

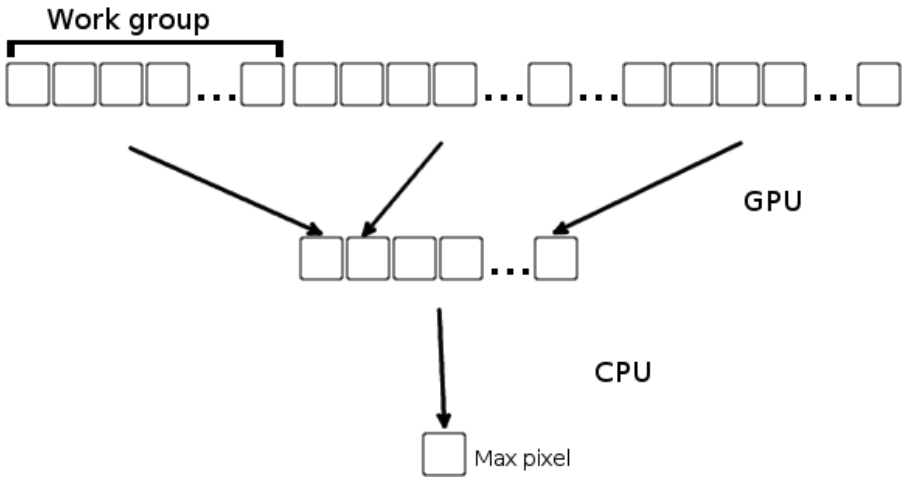

Figura 3.4: *Reducción en dos pasos.*

Algorithm 5 Reducción en dos etapas

Require: $Image$, $lines$, $samples$
Ensure: Posición del píxel más 'brillante'.

for $i = 0$ hasta $i < lines * samples/work\_group$ **do**
  $local\_max > Image[i]\ ?\ local\_max : Image[i]$
end for
for $i = 0$ hasta $i < work\_group$ **do**
  $local\_max[0] > local\_max[i]\ ?\ local\_max[0] : local\_max[i]$
end for
$global\_max[workgroup\_id] = local\_max[0]$
for $i = 0$ hasta $i < num\_work\_groups$ **do**
  $global\_max[0] > global\_max[i]\ ?\ global\_max[0] : global\_max[i]$
end for

tratado en *serie*. Si el error supera P_FA entonces se termina la ejecución, en caso contrario continuaríamos hasta llegar a $Nmax$ iteraciones.

3.2.2. Parallel Simplex Growing algorithm (P-SGA)

Una vez tenemos calculado el número de endmembers presentes en la imagen, gracias al algoritmo *GENE*, pasamos a la siguiente etapa, encontrarlos. Para ello aplicamos el algoritmo conocido como, *Simplex Growing Algorithm*[27]. Este algoritmo consiste básicamente en buscar los píxeles de la imagen de forma progresiva. Es decir, partiendo de los endmembers

encontrados anteriormente, se encuentra el siguiente. El algoritmo comienza inicialmente en un píxel cualquiera (*random*).

Una de las primeras decisiones que se tomaron fue como calcular el determinante de una matriz de tamaño variable. El tamaño máximo es $(n+1)*(n+1)$, donde n es el número de *endmembers*, se optó por usar la *Descomposición LU* (*Lower, Upper*):

$$Determinante(A) = Det(L) * Det(U) = Det(U) = \prod_{j=0}^{i+1} U_{jj}$$

Esto es porque el determinante L (*Lower*) tiene en su diagonal principal unos, y por tanto su determinante vale uno. Con cual para calcular el determinante de nuestra matiz debemos transformar en ceros los elementos que se encuentran por debajo de la diagonal principal (*matriz Upper*). Una vez terminado este paso, se procede a multiplicar los elementos de la diagonal principal para obtener el determinante. El tamaño de la matriz es variable como se indico anteriormente. El tamaño siempre vale:

$$(i+1)*(i+1) \qquad \forall \;\; 1 <= i < n.$$

Donde i se corresponde a la búsqueda del i-ésimo *endmember*.

Para formar la matriz, como ya hemos señalado, necesitamos los *endmembers* encontrados previamente (desde el primero hasta el i - 1). El primer *endmember* es seleccionado de manera aleatoria y sustituido por el real como primer endmember. La *imagen hiperespectral* tiene un tamaño de *samples*lines* píxeles de tal forma que para buscar un *endmember* tendremos que formar *samples*lines* matrices (3.5) y calcular el determinante de cada uno de ellos. Todas estas matrices tienen en común los *endmembers* encontrados anteriormente, es decir que gran parte de la matriz es común a todas las matrices que formaremos. Esto es muy importante ya que al usar *OpenCl* podremos paralelizar gran parte del calculo del determinante, ahorrando gran parte del computo y acelerando el algoritmo. Además el calculo de la zona común es compartido por todos los *threads*. Se paraleliza la construcción de la matriz así como su computo, en la versión serie esta operación se realiza *samples*lines*

veces, con éste método solo lo tendremos que hacer una vez por cada *endmember*.

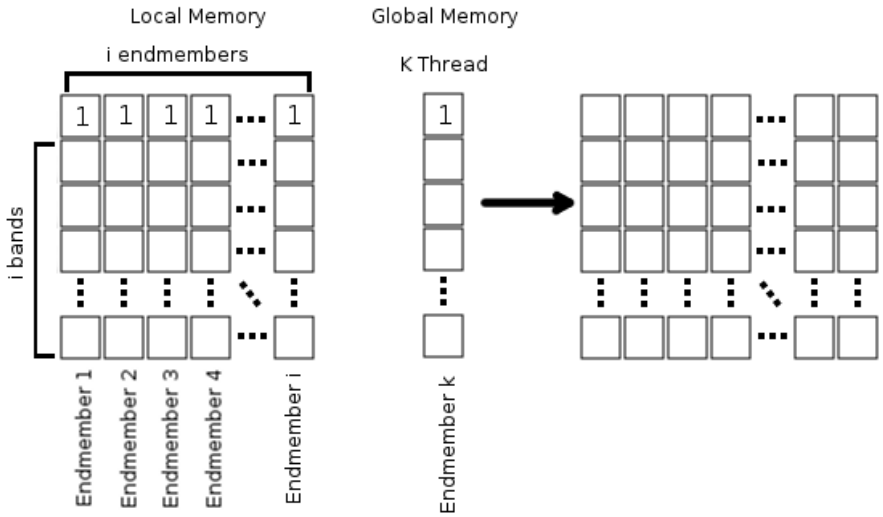

Figura 3.5: *Formación de la matriz para el calculo del i-esimo endmember*

Como se muestra en la figura (3.5) la formación de la matriz es bastante compleja, vamos a formar una matriz de tamaño $(i+1)*(i+1)$. De los cuales $i*(i+1)$ estarán almacenados en memoria local, cada hilo (**thread**) lleva un elemento de los *endmembers* anteriormente encontrados a **memoria local**, hasta formar una matriz parcial. Esta es la parte común a todas las matrices que se tienen que formar en la vuelta i-ésima del algoritmo. Al mismo tiempo cada hilo (*thread*) contendrá en memoria global un vector con el k-ésimo *endmember* (de tamaño i, donde i son las primeras bandas del píxel k). De tal forma que tenemos *samples*lines* hilos, cada uno de ellos con un posible *endmember* diferente, candidato a ser el i-ésimo *endmember*.

En la figura (3.6), se muestra como se realiza el proceso de diagonalización de la matriz. Transformando los elementos que se encuentran bajo la diagonal principal en ceros, de forma secuencial por filas. Como sabemos tenemos una gran parte de la matriz en memoria local (color verde), esta pueda ser modificada por distintos hilos, paralelamente. Al mismo tiempo cada hilo va modificar su propio vector (vector k-ésimo) almacenado en memoria global

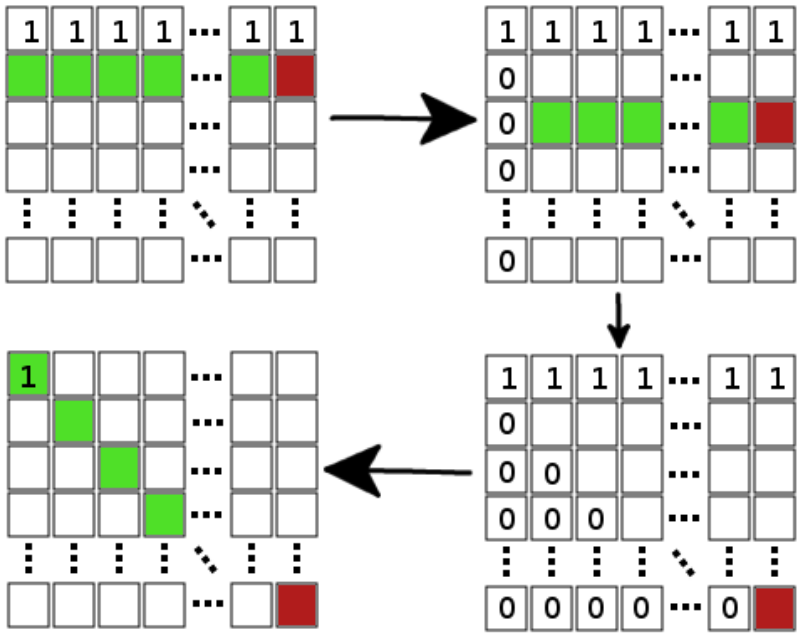

Figura 3.6: *Factorización LU, calculo del determinante a partir de la matriz superior.*

(color rojo). De tal forma que todos los hilos tienen acceso al resultado final (la *diagonal principal*) en memoria local y su parte del resultado final en memoria global (diferente en cada hilo):

$$Determinante(A)_k = (\prod_{j=0}^{i} U_{jj}) * k_{i+1}$$

El productorio es la parte de la diagonal principal de la matriz, común a todos los hilos que estén en *memoria local*. Cada hilo multiplicara este valor por el correspondiente en memoria global, calculado previamente. Por último el valor obtenido es dividido por el $factorial$ de i y calculado su valor absoluto. Todos los valores obtenidos son almacenados en una matriz de volúmenes transpuesta, de tamaño $(lines * samples)$.

En el proceso de reducción, figura (3.7), se tiene que buscar el máximo volumen entre todos los elementos calculados en el paso anterior. Está formado por dos etapas [29], en la primera se reduce de forma global, se divide la matriz de volúmenes en tamaño *work-*

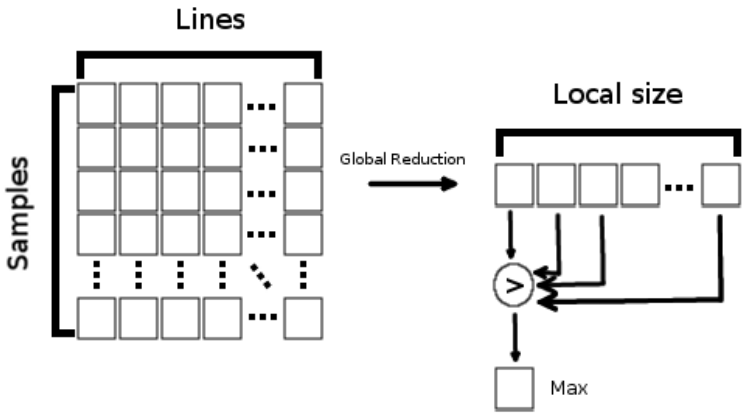

Figura 3.7: *Reducción de la matriz volúmenes.*

group-size. En cada grupo se buscará el máximo local, para después rellenar un vector de tamaño *samples*lines/work-group-size*, almacenado en memoria local. También se almacena la posición de los máximos locales, de tal forma que cuando se encuentre el máximo absoluto tendremos asociado su posición.

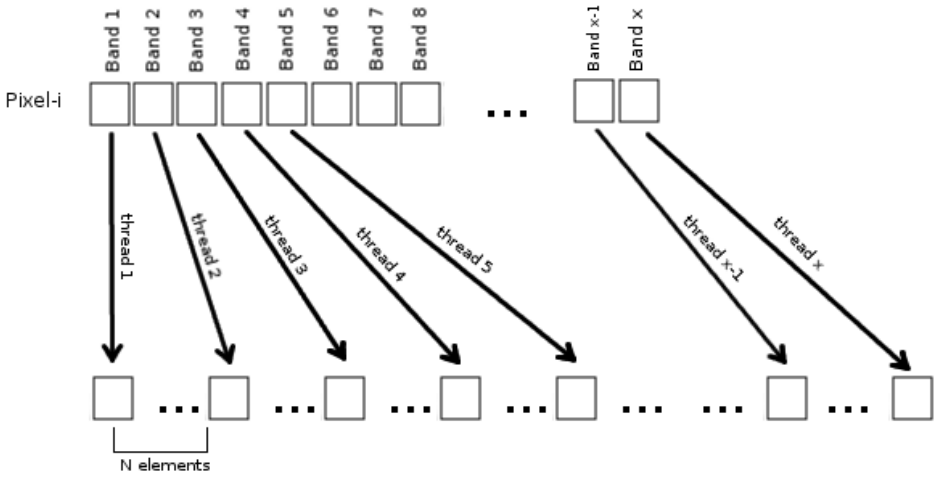

Figura 3.8: *Extracción del endmember, junto con todas sus bandas espectrales.*

Una vez tenemos la posición del píxel *endmember*, procedemos a extraer todas sus bandas y formar una matriz para la siguiente etapa, *SCLSU*. La forma de extracción es bastante

simple, cada hilo extrae un elemento y contamos con un número de hilos equivalente a las bandas que posea el píxel. Las posiciones están separadas N elementos, que se corresponde con el número de endmembers que tiene la imagen, porque *SCLSU* trabaja con la matriz transpuesta de endmembers.

Hay que destacar que todo este proceso tiene que ser llevado a cabo n veces, tantas como endmembers queremos encontrar. Por último, no por ello menos importante, debido a que usamos *doble precisión* para el calculo del determinante, aquellas imágenes que requieran buscar muchos endmembers (> 34) proporcionan como resultados valores incorrectos, este efecto provoca el desbordamiento y cálculo de valores incorrectos. Para solucionarlo basta con cambiar una parte del algoritmo. En vez de multiplicar toda la diagonal principal de la matriz, una vez convertida en matriz superior, se guarda el valor obtenido por cada hilo (correspondiente a la diagonal principal), y sin dividirlo por el factorial de i se tomará como volumen. Se ha conservado la estructura del algoritmo original para poder medir los tiempos con mas precisión, sin duda se podría acelerar más quitando dicha parte sin variar el resultado, como aparece en la siguiente expresión:

$$Determinante(A)_k = (\prod_{j=0}^{i} U_{jj}) * k_{i+1}$$

Una vez conseguimos una versión estable, se intentaron una serie de optimizaciones, siempre se ha procedido a compilar con la opción -$O3$ de compilación. Además se han comparado los tiempos de ejecución después de compilar con *gcc*, con los tiempos que ofrece compilar con *icc*. Esto se debe a que al trabajar en arquitecturas propias de *Intel*, a veces el código resulta más rápido con *icc*. En nuestro caso no se apreciaron mejoras significativas, no obstante es posible que para imágenes de mayor tamaño si sea influyente.

También se ha probado la técnica conocida como *padding* que consiste en alinear los objetos de memoria, de tal forma que al ser accedidos por diferentes hilos en *Device* sea más rápido, tampoco hemos apreciado mejora en los tiempos.

Por último la técnica de *auto-vectorización*, se resumen 4 pruebas, tabla (3.1). Desafor-

Compilador	optimización	opción de compilación
gcc	-O1	-fno-tree-vectorize -ffast-math
gcc	-O3	-ftree-vectorizer-verbose=6 -ffast-math
icc	-O1	-restrict -no-vec
icc	-O3	-restrict -axSSE4.1 -xSSE4.2 -opt-report

Tabla 3.1: *Pruebas auto-vectorización.*

tunadamente ninguna muestra cambios significativos, esto puede ser debido a que los datos no se encuentren consecutivos en memoria (esto es crítico para la vectorización).

También debemos señalar que se podrían mejorar los tiempos de ejecución, si en los *kernels* se manejasen datos en dos dimensiones (matrices) en vez de vectores. Cabe destacar que trabajamos con doble precisión, esto es debido a que los demás algoritmos, tanto *GENE* como *SCLSU*, necesitan usar doble precisión para dar resultados correctos. Este algoritmo por su parte no requiere doble precisión, dado que funciona correctamente con precisión simple.

3.2.3. Parallel Sum-to-one Constrained Linear Spectral Unmixing (P-SCLSU)

El último paso consiste en la construcción del mapa de abundancias a partir de la imagen hiperespectral original y los endmembers encontrados en el paso anterior (*SGA*). Para hacer un mapa con las abundancias, se necesita hacer una estimación de las abundancias de cada *endmember* en todos los píxeles de la imagen. Por cada *endmember* se crea un mapa de abundancias, es decir vamos a producir n mapas de abundancias, tantos como endmembers encontrados.

Con este algoritmo se ha querido probar, dos librerías para su aceleración. Partiendo de la base que ya se usaba la librería ***Blas*** (*Basic Linear Algebra Subprograms*) en *serie*, sobre todo el nivel tres (*Blas level 3*) que son multiplicaciones e inversiones de matrices ($\Theta(n^3)$). Se ha utilizado las librerías *ViennaCl* y *ClMagma*, para comprobar cual de las dos opciones ofrece un mejor rendimiento.

Lo primero que hay que destacar es que la librería ***Blas*** originalmente esta desarrollada en *Fortran*, el cual usa almacenamiento *column-major*, ordenamiento de la matriz por columnas. Esto es muy importante ya que *ViennaCl* usa *row-major*, ordenamiento natural de las matrices por filas. En general las matrices utilizadas no son cuadradas y por tanto la transpuesta de una matriz ordenada como *Row-major* no equivale a una matriz ordenada con *Column-major*. Resaltar dos aspectos a tener en cuenta antes de decidir usar las librerías: i) la penalización por transferencia de datos cuando usamos esta biblioteca y ii) la insuficiente extracción de paralelismo cuando usamos matrices de dimensiones reducidas. Experimentalmente en las *GPU* que hemos usado, dichas matrices deben ser mayores a $1000 * 1000$ elementos para que su cálculo obtenga mejoras al ser llevado a *GPU* (sin tener en cuenta el tiempo de transferencia).

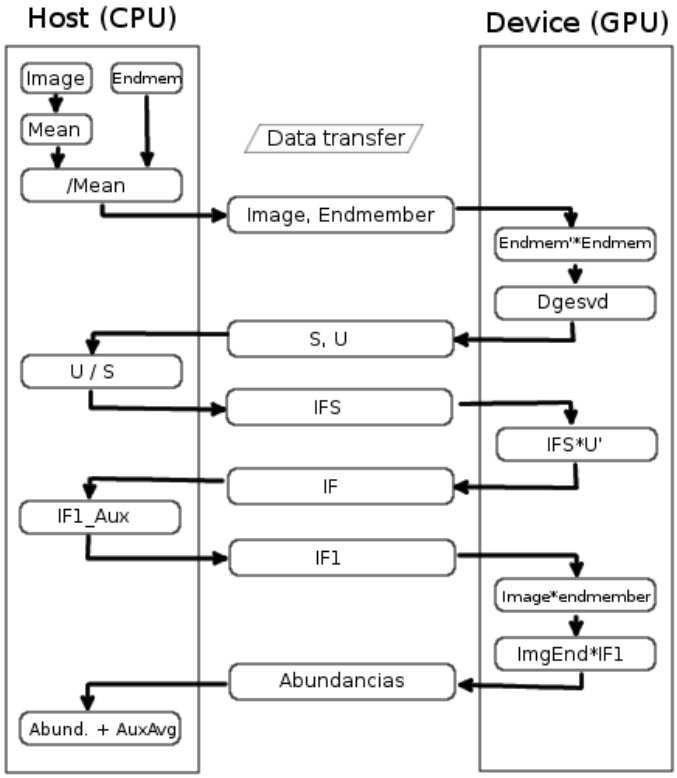

Figura 3.9: *Diagrama de flujo de la ejecución del algoritmo SCLSU*

Como se observa en la Figura (3.9), el flujo de ejecución del algoritmo conlleva una serie de transferencias para acelerar partes clave en el algoritmo. Hay que destacar la primera transferencia ya que es la de mayor tamaño, *image* tiene un tamaño *samples*lines*bands*, esto es de vital importancia en los resultados comparativos de ambas librerías. Previamente se hicieron pruebas sobre otra versión del algoritmo, debido a que usaba matrices más pequeñas y hacia un uso intensivo de transferencias, no se consiguió acelerar el código usando librerías.

$$Media = \sqrt[2]{\frac{1}{N} * \sum_{i=0}^{N} image_i^2} \qquad image = \frac{imagen}{media} \qquad endmember = \frac{endmember}{media}$$

Tenemos dos matrices de entrada, *imagen*, que es la imagen hiperespectral original, *endmember*, que es la matriz de endmembers encontrada por *SGA*. Lo primero que hacemos es calcular la media cuadrática sobre la *imagen* y a continuación dividimos ambas matrices por la media obtenida. Esta parte esta desarrollada en serie en la *CPU* ya que no hay ninguna función que nos facilite el calculo en *Device*.

$$dgesvd(endmember' * endmember) \qquad IFS * U'$$

A continuación multiplicamos la matriz transpuesta obtenida en el paso anterior por si misma, y a partir de ella usamos la función dgesvd. Esta función sirve para calcular la descomposición de valores singulares ($M = U * S * transpose(V)$). Donde se obtiene U, un vector de tamaño n, donde n son el número de endmembers, de valores singulares, V es la matriz de tamaño $n*n$ *ortogonal* (la matriz *inversa* coincide con su *transpuesta*), y un vector V que no se va utilizar en nuestro caso. Es entonces cuando se divide la matriz U entre el vector S para obtener una matriz que llamaremos IFS esto se realiza en serie debido al tamaño de las matrices implicadas. El resultado lo llevamos a *Device* donde se multiplicará por U'. con lo obtenido se calcula un vector auxiliar con la media por filas, *auxAvg*, y una matriz corregida con la media calculada en *auxAvg* a esta matriz la llamaremos *IF1*.

$$(image^{*}endmember) * IF1) + auxAvg$$

Como último paso para la reconstrucción de la matriz abundancias, multiplicamos las matrices *imagen* por *endmember*, al resultado es multiplicado por *IF1* todo ello se lleva a cabo en *Device*. El resultado final se lleva a *Host* y se le suma la media obtenida anteriormente que llamamos *auxAvg* por filas.

A lo largo de este capítulo tres, se ha abordado una introducción al paradigma de programación paralela, posteriormente se ha mostrado las implementaciones llevadas a cabo para cada uno de los tres algoritmos considerados en la cadena de desmezclado: *Parallel Geometry-based Estimation of Number of Endmembers* (**P-GENE**), *Parallel Simplex Growing Algorithm* (**P-SGA**) y *Parallel Sum-to-one Constrained Linear Spectral Unmixing* (**P-SCLSU**).

Capítulo 4

Resultados

4.1. Imágenes. Sintéticas vs Reales

Las imágenes *hiperespectrales* reales, son una colección de datos recogidos por sensores en longitudes de onda que varían entre 0,4μm y 2,5μm, agrupadas por bandas espectrales, sobre una misma zona. Mientras que las imágenes *sintéticas* están creadas por un algoritmo que parte de la base de una serie de firmas espectrales reales, con ellas se crea un mapa de abundancias para cada píxel, pudiendo elegir la cantidad de *endmembers* que compondrán la imagen. Las imágenes usadas durante este proyecto pueden verse en la tabla (4.1), el tamaño que se indica corresponde con samples*lines*bands, se han marcado en negrita las imágenes que se han usado para presentar resultados, también se indica el tiempo-real de procesamiento en las imágenes que se han utilizado.

Name	Dimensions	Size	Num. endmembers	Real-time
Cuprite	$350 * 350 * 188$	$43,9MB$	19	$1,98s$
SubsetWTC	$512 * 614 * 224$	$134,3MB$	31	$5,09s$
Sintetica2	$750 * 650 * 224$	$416,5MB$	30	$7,88s$

Tabla 4.1: *Características de las imágenes utilizadas en el proyecto.*

La figura (4.1) muestra la imagen *Cuprite*, la cual está tomada por el sensor *AVIRIS*, sobre una zona rica en minerales en la región de Nevada, Estados Unidos. Esta imagen ha sido ampliamente utilizada para los estudios de desmezclado hiperespectral por que se

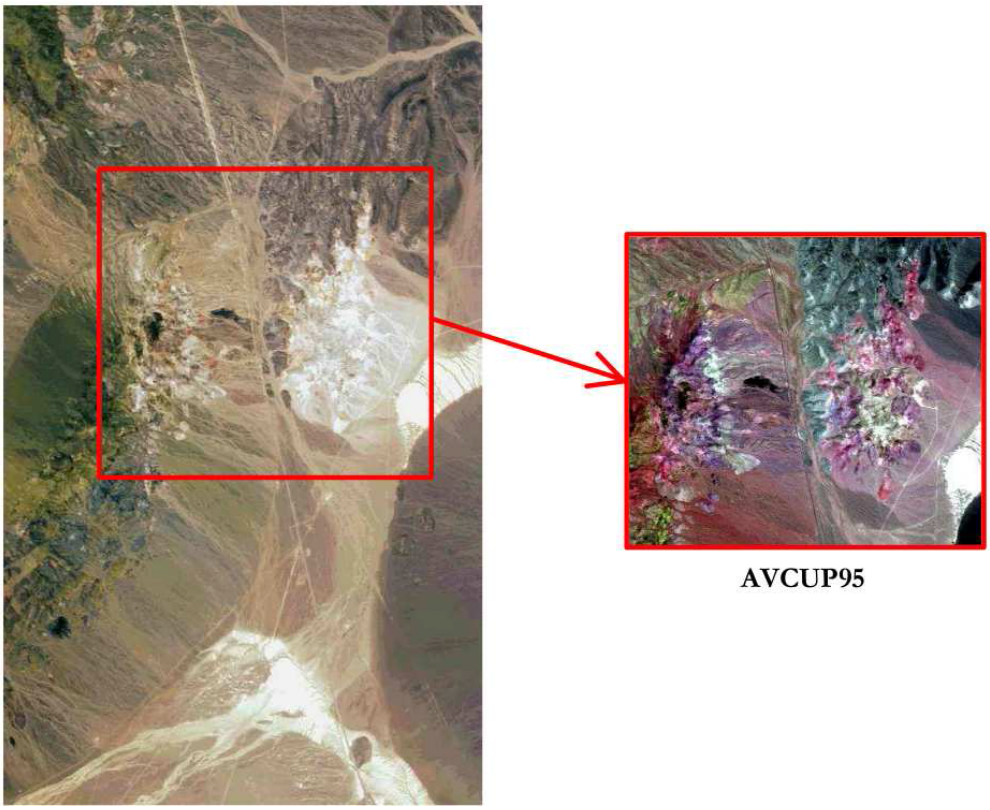

Figura 4.1: *Imagen de Cuprite sobre una fotografía área de alta resolución.*

dispone de datos de referencia. La resolución es de 20m/píxel.

En la figura (4.1) podemos visualizar la ubicación de la imagen Cuprite, sobre una fotografía aérea de alta resolución.

En la figura (4.2) se puede apreciar la imagen *SubsetWTC*, es una imagen tomada también por el sensor *AVIRIS* unos días después de los atentados del *World Trade Center*, la resolución es de 1,7m/píxel ya que fue tomada a menor altura, la finalidad de esta imagen fue obtener datos del fuego ocasionado por los atentados.

En la figura (4.2) podemos ver una composición en falso color de la imagen hiperespectral obtenida cinco días después de los ataques terroristas sobre el World Trade Center. El recuadro rojo marca la zona en la que se encontraban las torres.

Figura 4.2: *Composición en falso color de la imagen SubsetWTC.*

4.2. Plataformas heterogéneas

Una de las características que hace atractivo el uso de *OpenCl*, es la portabilidad de código en un amplio espectro de plataformas de distinta finalidad, desde *CPU's* hasta *GPU's* pasando por *FPGA's*, *DSP's* o *Aceleradores*.

En la tabla (4.2) tenemos las principales características de las plataformas que hemos

	CPU	GPU	Acelerador
Name	Xeon E5-2695 v3	Geforce Gtx 980	Xeon Phi 31S1P
OpenCl Version	1.2	1.2	1.2
Frecuency	2.3 Ghz	1.2Ghz	1.1 Ghz
Max compute units	56	16	224
Local memory size	32 KB	48 KB	32 KB
Global memory size	64 GB	4 GB	5,6 GB
Constant Memory size	128 KB	64 KB	128 KB
Max. work group size	8192	1024	8192
Max. work item dimension	3	3	3
Max. work item sizes	8192*8192*8192	1024*1024*64	8192*8192*8192

Tabla 4.2: *Características CPU Xeon, GPU NVidia, Acelerador Xeon Phi*

empleado en este proyecto.

4.2.1. CPU Xeon

La unidad central de proceso (*CPU*) es la parte del hardware encargada de la ejecución del programa. De propósito general pero también es la encargada en repartir el trabajo entre otros co-procesadores (aceleradoras, gráficas etc.). Tiene una jerarquía de memoria, que mantiene coherencia en los datos y permite a la CPU trabajar a una mayor frecuencia. La jerarquía de memoria puede resumirse de la siguiente forma: en un primer nivel, Memoria secundaria, disco duro y/o disco de estado solido (*SSD*), en un segundo nivel tenemos la memoria *RAM* (volátil), y como tercer nivel, la memoria *caché*, la más rápida y cercana a la CPU. Esta última se encuentra en el interior del núcleo y representa una gran parte de la superficie total del mismo.

Como hemos señalado anteriormente, se han incluido algunas características destacables de **Intel Xeon E5-2695**[1] en la tabla (4.2). En nuestro caso disponemos de 16 núcleos, 2 hilos por núcleo, tiene soporte para memorias ECC, Error Code Correction, para corregir tipos comunes de corrupciones de datos.

[1]http://ark.intel.com/es-es/products/81057/Intel-Xeon-Processor-E5-2695-v3-35M-Cache-2_30-GHz

4.2.2. GPU NVidia

En los últimos años el crecimiento en la potencia de las unidades de procesamiento gráfico (GPU's), ha sido gracias, en gran medida, por el sector de los videojuegos. Este hecho nos han ofrecido un gran avance en el campo de la paralelización de algoritmos ya que esta plataforma se caracteriza por tener múltiples unidades de procesamiento simplificados. Estas pequeñas unidades realizan una misma tarea sobre datos que no tengan dependencias entre si, típicamente imágenes. Dicha especialización queda plasmada en el uso de calculo en coma flotante. La *GPU* trabaja con una frecuencia de reloj menor a las CPU's, entorno a 1,2-1,5 Ghz y posee su propia jerarquía de memoria. Las unidades de procesamiento están agrupadas en (*warp*) de tal forma que comparten una memoria caché de constantes y una pequeña memoria (que llamaremos memoria local), típicamente en arquitecturas NVidia están agrupados en tamaños de 32 mientras que en AMD es de 64. Todos los hilos del mismo *warp* se ejecutan de manera simultanea, concurrente, entre *warps* las ejecuciones son independientes y todos ellos acceden a una memoria global común, *GDDR5*.

Podemos visualizar las características destacables de **Nvidia Geforce Gtx 980**[2] en la tabla (4.2). También podemos ver en la figura (4.3)[3] la arquitectura de nuestra gráfica utilizada en los test.

Para entender mejor el concepto de hilos (*threads*), sus ejecuciones y dependencias. Vamos a utilizar una analogía. Digamos que tenemos 10240 alumnos para evaluar, cada facultad tiene un máximo de 1024 alumnos (*max work group size*) y que hasta que no termine la facultad A no empieza a dar clases la facultad B. Cada facultad esta dividida en 32 clases (*warp*) de 32 alumnos cada una. Las clases son independientes entre si, pero dan las mismas materias. Por ejemplo, podemos poner de acuerdo dos facultades para que cada una atienda a 512 alumnos cada una (max work item size), 512*2*1 o distribuirlos en 7 clases en cada facultad divididas en dos plantas, repartidos en 4 facultades, 128*4*2. Cualquier combinación

[2]http://www.nvidia.es/object/geforce-gtx-980-es.html
[3]http://www.nvidia.es/object/maxwell-gpu-architecture-es.html

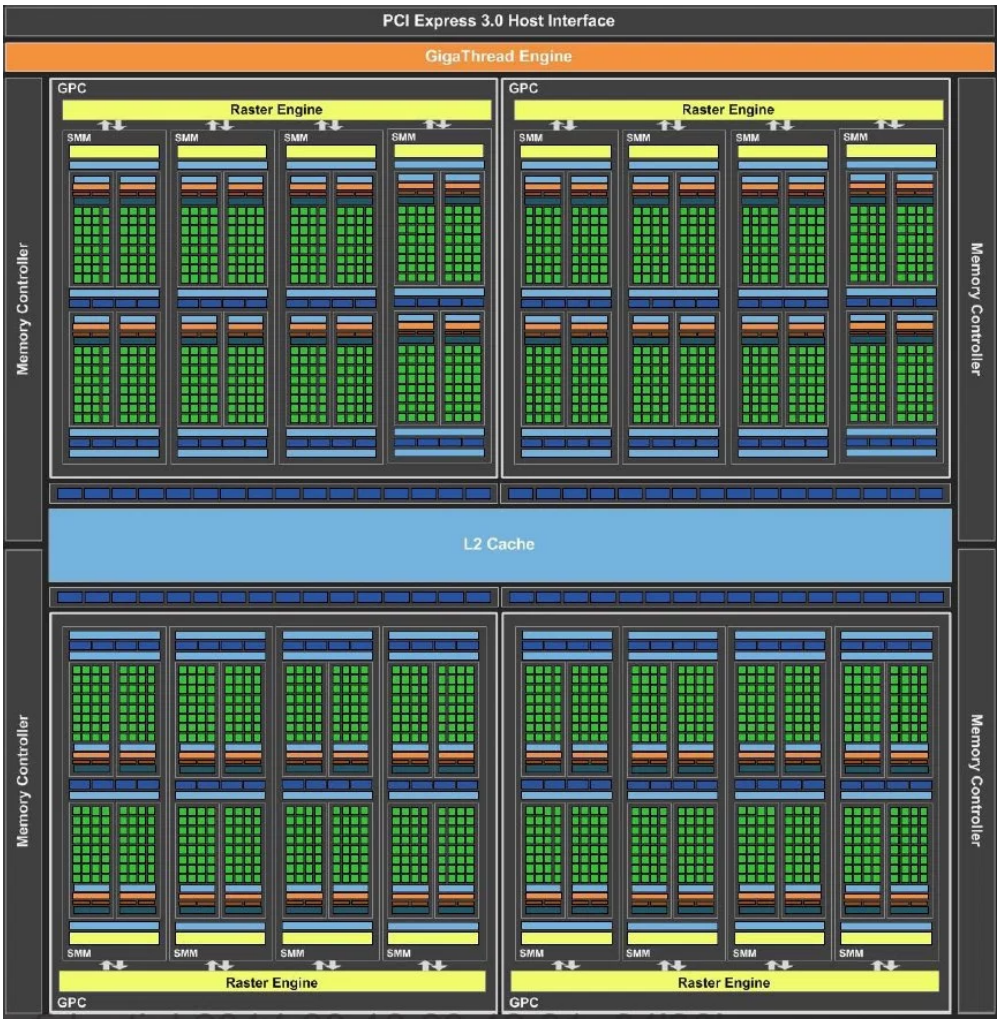

Figura 4.3: *Estructura de la arquitectura Maxwell de NVidia.*

que usemos nunca puede superar max work group size, ni superar las dimensiones *max work item size*, por ejemplo un facultad que tenga más de 64 plantas. Siguiendo esta analogía la memoria local seria la información que tenemos en la pizarra, solo los alumnos de esa clase pueden consultarla. La memoria global seria la biblioteca todos los alumnos pueden consultarla pero tardan más en acceder a ella (está en otro edificio).

La tarjeta en la que se ha probado los algoritmos tiene una arquitectura **Maxwell**, las características destacadas frente a la anterior generación son: un menor consumo, como consecuencia de ampliación de caché y reducción del bus de memoria a 128 bits, entre otros.

La próxima arquitectura, llamada **Pascal**[4], promete tener memoria unificada entre CPU y GPU, nueva unidad de precisión, *half precision point fp16*, entre otros.

4.2.3. Acelerador Xeon Phi

El co-procesador Intel Xeon Phi, esta basado en la arquitectura MIC (*Many Integrated Cores*), en nuestro caso tenemos 57 núcleos con 4 hilos por nucleo (*Hyperthreading*). Están conectados por un anillo bidireccional que proporciona una interconexión de gran velocidad. Cada núcleo dispone de una *caché* privada como muestra la figura (4.4)[5].

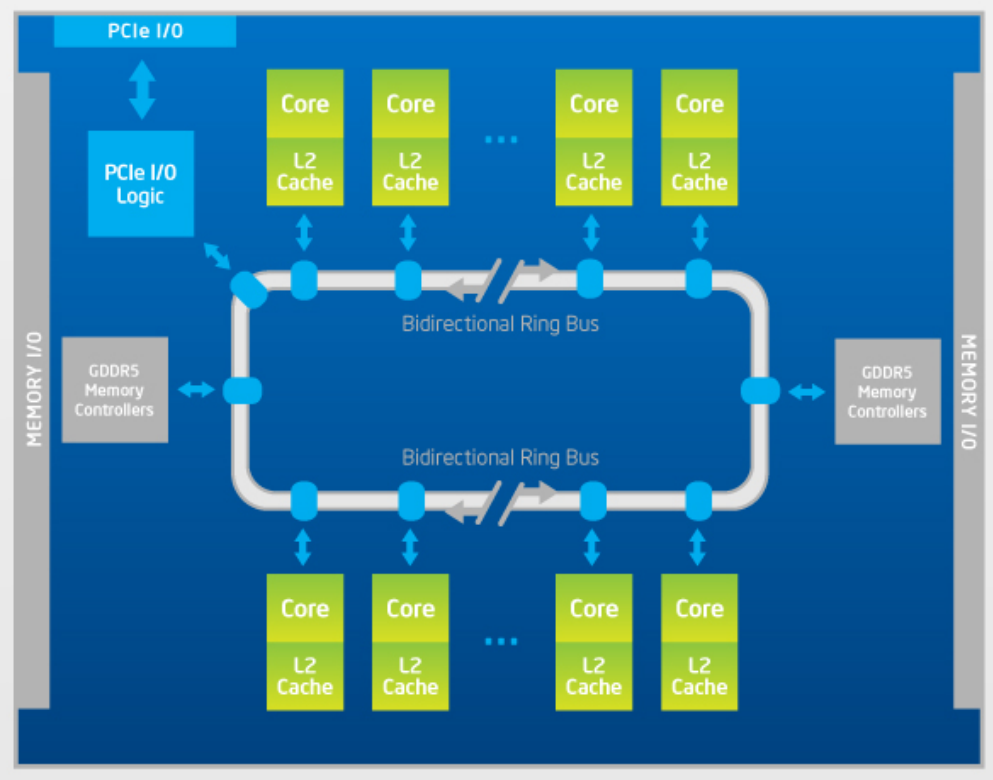

Figura 4.4: *Arquitectura en anillo de los Intel Xeon Phi.*

Se han incluido algunas características destacables de **Intel Xeon Phi 31S1P**[6] en la

[4]http://www.nvidia.com/object/gpu-architecture.html
[5]http://www.intel.es/content/www/es/es/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html
[6]http://ark.intel.com/es-es/products/79539/Intel-Xeon-Phi-Coprocessor-31S1P-8GB-1_100-GHz-57-core

tabla (4.2). Una diferencia de las arquitecturas Aceleradoras frente a las *GPU* es la cantidad de chips dedicada a memoria caché tanto L1 como L2, (las *GPU* dedica la mayor parte a tener muchos cores) esto es similar a las arquitecturas *CPU*. Al igual que las *CPU* en la jerarquía de memorias de las Xeon Phi mantiene la coherencia entre memorias, el tipo de memoria que utiliza es *GDDR5*, comúnmente usadas en *GPU*'s. Cada núcleo posee una unidad vectorial que soporta instrucciones de 512 bits, esto reduce mucho el consumo ya que una instrucción es capaz de procesar más datos (SIMD).

4.3. Bibliotecas

En esta sección se van a destacar las cualidades de las librerías usadas, *ViennaCl*[7] y *ClMagma*[8]. Ambas librerías están dedicadas a la aceleración de los algoritmos conocidos como **Blas** bajo el paradigma de programación *OpenCl*, en general se trata de operaciones sobre matrices, producto, calculo de inversas etc. Se ha trabajado con la versión 1.7.1 de *ViennaCl* y 1.3.0 en el caso de *ClMagma*.

Sobre *ViennaCl* queremos destacar que es muy rápido en el computo de las operaciones, del orden 3x más rápido que *ClMagma*. Hay que tener en cuenta que las operaciones que maneja es en dos dimensiones, es decir matrices, esto supone un cambio respecto a *BLAS* que solo usa vectores (matrices *1D*), el problema surge a la hora de transferir los datos desde *Host* a *Device* y viceversa. En la documentación que acompaña a la librería aconseja usar la librería *Boost-Ublas*[9] para transferir las matrices. La transferencia de datos penaliza hasta el punto que no compensa la aceleración obtenida por su uso. Otra opción que se ha encontrado es usar matrices de la forma $vector < vector < double >>$, que aunque es algo más rápida que la anterior, no es suficiente para competir con la librería *ClMagma*. Así pues en general podemos afirmar que a la hora de operar no tiene rival, no al menos con la otra librería probada, la transferencia de matrices supone un problema a la hora de

[7] http://viennacl.sourceforge.net/
[8] http://icl.cs.utk.edu/magma/software/view.html?id=190
[9] http://www.boost.org/doc/libs/1_61_0/libs/numeric/ublas/doc/

acelerar el algoritmo. También queremos destacar la facilidad en el uso de las funciones, gracias a la *sobrecarga* de operadores y funciones, se facilita la claridad en el código, *por ejemplo* podemos escribir $vectorC = vectorA + vectorB$, siendo los tres vectores de la misma longitud se sumaran elemento a elemento. Por último como se comento anteriormente las operaciones son *Row-major*, ordenación por filas, que es como calculamos de manera natural el producto de matrices por ejemplo.

En el caso de *ClMagma* es una traducción más literal de la librería *Blas*. Los nombres de las funciones, los parámetros que usan las funciones siguen el mismo patrón, lo cual facilita mucho a la hora de traducir las funciones. También utiliza la ordenación en memoria *Column-major*, ordenación en columnas, que puede suponer un desafío a la hora de comprobar los resultados. Por otro lado las transferencias, al manejar matrices en forma $1D$, es decir vectores, son muy rápidas y por consiguiente tienen un rendimiento muy aceptable. Como se apuntaba anteriormente el rendimiento que tienen las operaciones es algo peor que *ViennaCl*. Otro punto a favor es la gran comunidad que esta desarrollando esta librería, ya que tuvimos un obstáculo en el desarrollo de *GENE*, y usuarios de esta comunidad nos proporcionaron una solución a la duda de implementación en muy poco tiempo (se facilitará en próximas versiones una función para evitar dicho problema).

Una posibilidad que se podría plantear es utilizar las transferencias de ClMagma y el computo de *ViennaCl*, pero esto no es posible ya que esta última librería necesita tener los datos en una estructura $2D$, matrices.

En el caso de trabajar con estructuras de datos pequeñas, ambas librerías obtienen un rendimiento escaso, siendo negativo en ciertos casos operar en *Device*, por ello conviene hacer comparativas en tiempos siempre que se utilice cualquiera de las dos librerías. En nuestro caso hemos observado que con las plataformas que hemos probado, realizar multiplicaciones entre matrices de tamaño menor a $1000 * 1000$ es más recomendable en *serie*. Ambas librerías proporcionan funciones para medir el tiempo de ejecución, *timer()* en el caso de *ViennaCl* y *magma_sync_wtime()* en el de *ClMagma*, ambas basados en *gettimeofday()* de **POSIX**.

4.4. Métricas

En esta sección hablaremos de las métricas que comparan la precisión de la cadena de desmezclado propuesta. En concreto vamos a medir la calidad del código desde dos puntos de vista. Primero la calidad de los resultados, para ello cada uno de los algoritmos tiene asociado una metodología para comprobar como son los resultados ya sea respecto a su versión serie (SAD) o al error producido ($RMSE$). La segunda métrica tiene en cuenta la mejora experimentada respecto a su versión serie.

4.4.1. Calidad

A la hora de medir la calidad de los resultados obtenidos no solo se han tenido en cuenta la obtención de los mismos resultados que en *serie* si no que además cada algoritmo tiene sus propias métricas que miden el error producido al ejecutar el algoritmo en concreto. Las pruebas se han realizado sobre las imágenes *Cuprite*, *SubsetWTC* y una imagen *Sintética*.

Para el algoritmo *GENE*, lo que se busca es obtener unos resultados similares a los esperados en cuanto a cantidad de *endmembers* a encontrar, con probabilidad de falsa alarma baja. Para ello lo que se prueba son distintas configuraciones hasta obtener una tabla con la que comparar. En nuestro caso conocemos previamente los resultados para dos de las imágenes, *Cuprite* que gracias a las mediciones de la *United States Geological Survey* (*USGS*) conocemos tanto los minerales (firmas espectrales) como la cantidad de *endmembers* que podemos encontrar. También usamos para el estudio una imagen hiperespectral *sintética*, debido a su naturaleza sabemos con cuantos *endmembers* fue creado.

En el caso del algoritmo *SGA*, se calcula como el angulo que forman dos firmas espectrales (*Spectral Angle Distance*). Dicho angulo puede variar entre 0º y 90º, donde 0º es un una buena firma espectral y 90º muy mala firma espectral respecto a las muestras comparadas. Las muestras espectrales con las que se van a comparar los endmembers encontrados por *SGA*, están disponibles *online*[10], llamaremos X a un píxel entre los extraídos por el algoritmo

[10]https://www.usgs.gov/

e Y a los datos extraídos por *USGS*, ambos contienen el mismo número de bandas, L [14]:

$$SAD(X,Y) = Arcos \frac{\sum_{i=0}^{L-1} X_i * Y_i}{\sqrt[2]{\sum_{i=0}^{L-1} X_i^2} * \sqrt[2]{\sum_{i=0}^{L-1} Y_i^2}}$$

Se saben las firmas espectrales de cada uno de los minerales que se pueden encontrar. Para cada uno se compara con todos los endmembers encontrados hasta dar con el más parecido, y entonces se toma el valor de la formula anterior descrita.

Por último, para medir la calidad del algoritmo *SCLSU*, se utiliza la medida conocida como *Media cuadrática del error* (*RMSE*), mide el error producido después de reconstruir la imagen de abundancias, X, cuando es multiplicada por la matriz de endmembers identificados, M, que llamaremos, $\hat{Y}$, respecto a la imagen original, Y:

$$RMSE(Y,\hat{Y}) = \frac{1}{N} \sum_{i=0}^{N} (\sum_{j=0}^{L} (y_i^j - \hat{y}_i^j)^2)^{1/2}$$

donde $0 <= N < lines * samples$ y $0 <= L < bands$ [14].

4.4.2. Rendimiento

Para medir el rendimiento se ha tenido en cuenta la aceleración de los algoritmos respecto a la versión serie, además, como es el objetivo, se ha identificado cuanto es el tiempo límite, conocido como *tiempo-real*, para el procesamiento de la cadena completa de desmezclado. Hay que destacar que se ha trabajado con imágenes relativamente pequeñas pero sabemos que si conseguimos que estas imágenes se puedan procesar en *tiempo-real*, imágenes de mayor tamaño también se podrán procesar. Esto se debe a que el aumento en el tiempo respecto al tamaño de la imagen es logarítmico, es decir aumenta poco ante variaciones en tamaño de la imagen grandes (y/o del número de endmembers).

$$Rendimiento = \frac{Tiempo \quad serie}{Tiempo \quad OpenCl}$$

La metodología usada para medir los tiempos ha sido la siguiente, se toman 10 medidas de las cuales se elimina el mejor y el peor tiempo, y se calcula la media. El compilador usado en estas medidas ha sido *gcc v.4.9.2*. En ambos casos (*serie y OpenCl*) se ha usado el flag -*O*3. También se han considerado la toma de tiempos con *icc v.16.0.1*, pero los resultados no están aquí mostrados debido a la similitud obtenida respecto al otro compilador. También destacar que a la hora de medir los tiempos, *SGA* y *SCLSU* se han medido 19 endmembers en el caso de *Cuprite* y 30 en el caso de la *Sintética*, para *GENE* los valores de entrada de *número máximo de endmembers* han sido de 30 en el caso de *Cuprite* y 40 en el caso de la imagen *Sintética*, así como la *probabilidad de falsa alarma* fue de 0 y 10^{-3} respectivamente. Tanto en los tiempos de la cadena completa (*GENE+SGA+SCLSU*) como en la cadena parcial (*GENE+SCLSU*) se ha partido de los datos de entrada de *GENE* mencionados anteriormente. Se ha hecho esta distinción entre algoritmos y cadenas para así dar más datos de la evolución de la aceleración. En el caso de la imagen *SubsetWTC*, solo son mostrados los resultados de la cadena completa, se han partido de 42 como *máximo numero de endmembers* y 0 probabilidad de falsa alarma (se encuentran 39 *endmembers*), para tomar los tiempos de la cadena completa. Por último destacar que los tiempos en *serie* son tomados con un solo hilo de ejecución.

4.5. Resultados experimentales

4.5.1. Calidad

Como se aprecia en la tabla (4.3), hemos probado diversas combinaciones con los dos parámetros de entrada del algoritmo *GENE*. Hemos usado las imágenes de *Cuprite* y una *Sintética*, se sabe que tienen 19 y 30 endmembers respectivamente, hay que destacar que se obtienen los mismos resultados que la versión serie. Con *Cuprite* se obtienen 27 *endmembers* como valor más cercano al real, destacar que este valor es muy alto pero es similar a otros algoritmos que realizan la misma función, como es el popular *virtual dimensionality* (VD, $\hat{p} = 28$ prob. falsa alarma $= 10^{-2}$) o HySime ($\hat{p} = 16$). Respecto a la imagen *Sintética* se

ha elegido el par 40 máximo numero de endmembers a encontrar y la menor probabilidad de falsa alarma, 10^{-3}. Como combinación más cercana a la realidad, sabemos que tiene 30 endmembers porque es lo que se había especificado a la hora de ser creado.

Max endmembers	Prob. falsa alarma	Cuprite $\hat{p}$	Sintética $\hat{p}$
25	$10^{-1} \dots 10^{-8}$	25	25
25	0	25	25
30	10^{-1}	30	30
30	$10^{-2} \dots 10^{-8}$	30	28
30	0	**27**	28
35	10^{-1}	35	30
35	$10^{-2} \dots 10^{-7}$	35	29
35	$10^{-8} \dots 0$	32	29
40	10^{-1}	37	30
40	$10^{-2} \dots 10^{-3}$	36	**30**
40	$10^{-4} \dots 0$	36	29
45	10^{-1}	43	29
45	10^{-2}	41	29
45	$10^{-3} \dots 10^{-6}$	40	29
45	$10^{-7} \dots 10^{-8}$	39	29
45	0	38	29

Tabla 4.3: *Ejecuciones con diferentes valores de entrada P_FA y max-endmembers.*

En el caso del valor del ángulo espectral, tabla (4.4), solo disponemos de datos reales de la imagen *Cuprite*, así pues es con estos con los que se han calculado el ángulo. Hay 5 minerales presentes en la zona de *Cuprite*, la *Buddingtonita* y la *Moscovita* son las muestras que mejor recogen los *endmembers* respecto a la realidad. Estos ángulos fueron calculados respecto a los 19 *endmembers* obtenidos por *SGA*. En caso de utilizar la cadena completa y encontrar 27 *endmembers* es posible que el ángulo de algunos minerales sea menor ya que tendría mas endmembers con los que comparar.

4.5.2. Rendimiento

En primer lugar vamos a medir el rendimiento respecto a la versión *serie* para cada algoritmo. A continuación mediremos el rendimiento de dos cadenas, *GENE+SCLSU* que

Alunita	Buddingtonita	Calcita	Caolinita	Moscovita	Media
8,20º	4,17º	5,87º	10,17º	5,41º	6,76º

Tabla 4.4: *Valores del ángulo espectral (en grados) de los endmembers encontrados respecto a los endmembers conocidos en el área de Cuprite.*

Cuprite	Tiempo (s)	Speedup	Host	Transfer	Device
Serie	2,688	-	0,00 %	0,00 %	100,00 %
OpenCl CPU	42,797	-	3,45 %	2,18 %	94,4 %
OpenCl GPU	0,845	3,18x	12,00 %	3,80 %	**84,20** %
OpenCl Acelerador	792,058	-	0,00 %	0,00 %	100,00 %
Sintética	Tiempo (s)	Speedup	Host	Transfer	Device
Serie	27,595	-	0.00 %	0.00 %	100.00 %
OpenCl GPU	4,298	6,42x	4,00 %	2,50 %	**93,50** %
OpenCl Acelerador	828,484	-	0,00 %	0,00 %	100,00 %

Tabla 4.5: *Tiempos y aceleraciones en diferentes plataformas para el algoritmo GENE.*

aunque no es la cadena completa, nos sirve para generar unos mapas de abundancias para los endmembers encontrados por *GENE*. La cadena completa *GENE+SGA+SCLSU* es importante compararla con el tiempo-real de cada imagen que tenemos en la tabla (4.1). Destacar también que en el caso de los algoritmos *SGA* y *SCLSU* los tiempos obtenidos han variado muy poco en el caso de la plataforma *GPU*, del orden de varias milésimas.

En primer lugar el algoritmo *GENE*, tabla (4.5), tenemos los tiempos respecto al algoritmo en *serie*. Comentar que en *serie* no existe concepto de *Device* (pertenece al ámbito *OpenCl*), es solo con motivo ilustrativo para poder comparar mejor. El algoritmo *GENE* se caracteriza por tener gran parte de código escrito gracias a la librería *Blas*. Tiene partes en serie que se pueden optimizar, estas fueron aceleradas con un *kernel OpenCl*. Lo más destacable de la versión *GPU* es que gran parte del computo se realiza en paralelo (ya sea gracias a *ClMagma* o a un *kernel*), aunque también tiene parte del código en serie. La aceleración aunque discreta $3,18x$ y $6,42x$, se ha podido comparar con una versión acelerada con *CUDA* y los resultados son muy similares en ambas imágenes, $3{,}63x$ en el caso de *Cuprite* se consigue gracias a *CUDA*. En caso de usar la librería *ClMagma* en otras plataformas ya

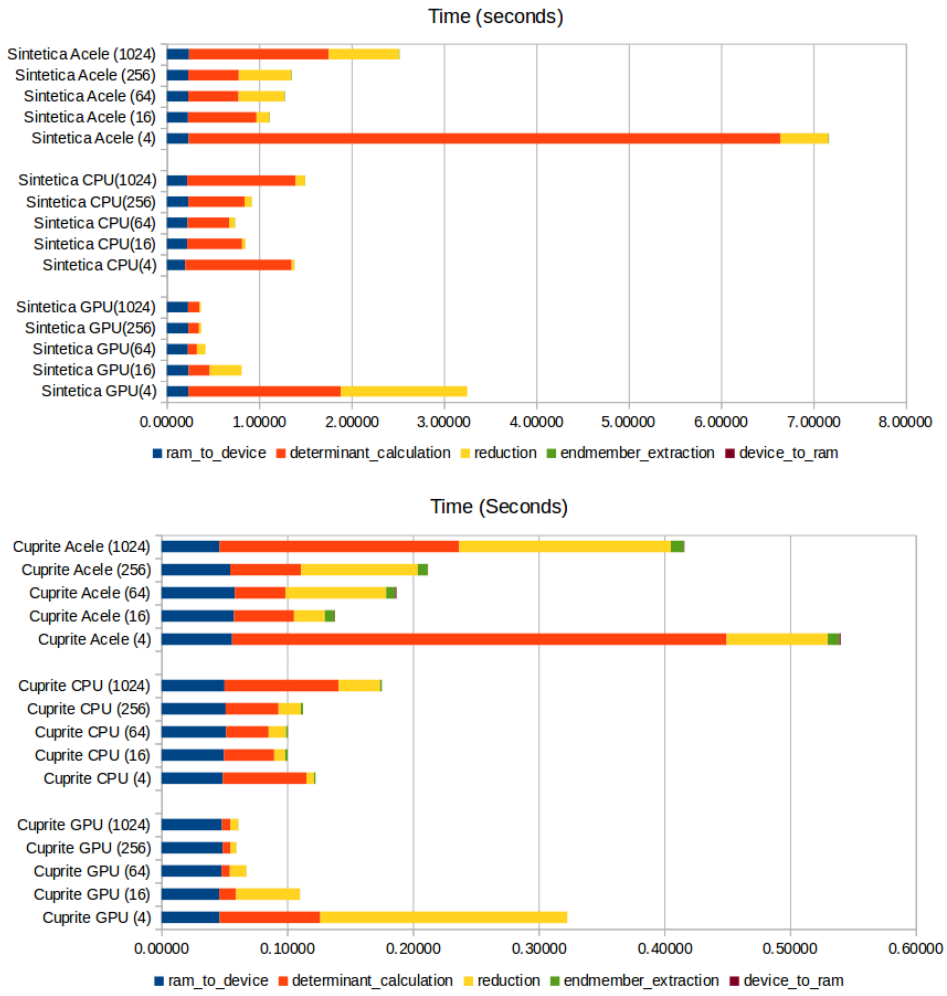

Figura 4.5: *Comparativa de tiempos para diferentes local size y diferentes plataformas para el algoritmo SGA.*

sea *CPU* o *Xeon-Phi* se obtienen resultados muy desfavorables. Es posible que esta librería este solo optimizada para arquitecturas tipo *GPU*.

En el caso de *SGA* hay que destacar que la versión *serie* fue traducida directamente de **Matlab** y por tanto puede no estar todo lo optimizado que seria deseable, ya que nuestra tarea ha sido acelerar la versión *OpenCl*. Sin entrar en detalles la optimización de *OpenCl* es fundamentalmente debido a la paralelización del cálculo del determinante, esto no es posible aplicarlo a la versión *serie* por dependencias en su cálculo. En primer lugar se ha incluido,

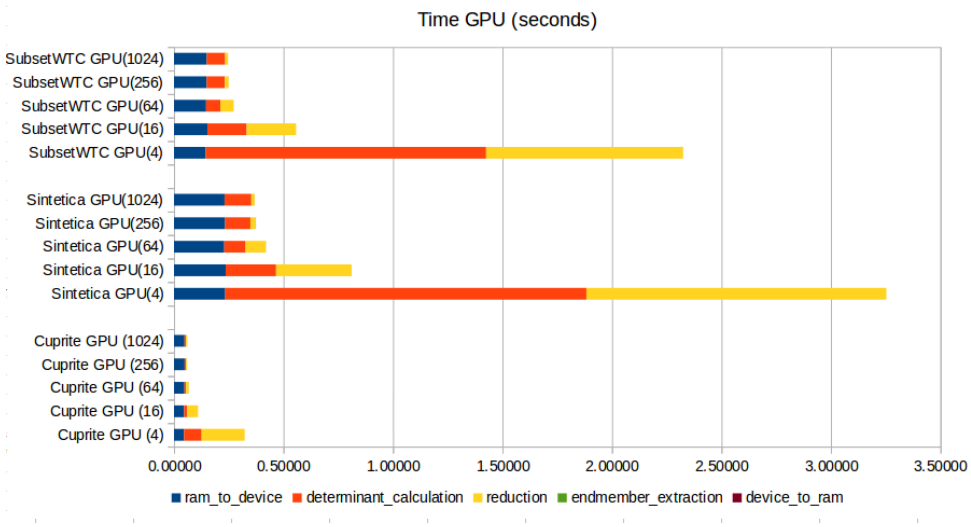

Figura 4.6: *Comparativa de tiempos de diferentes imágenes y local size en GPU.*

entre paréntesis, los tamaños de *local size* óptimos en cuanto a tiempo se refiere, destacar que en caso de dejar al compilador encontrar el tamaño optimo de *local size*, mejora ligeramente los tiempos, pero este parámetro fue introducido para poder hacer comparativas. En todas las imágenes se han calculado los tamaños de *local size* potencia de cuatro (4, 16, 64, 256 y 1024). Lo que observamos en la figura (4.5) es comparativa de las dos imágenes, *Cuprite* y *Sintética*, observamos en el caso de *GPU* siempre se mejora los tiempos aumentando el tamaño de *local size*, aunque a partir de 256 los cambios son muy pequeños. En el caso de *CPU* y *Aceleradora* tenemos unos tamaños mejores, 64 y 16 respectivamente, a partir del cual el rendimiento empeora. En el caso de las tres *imágenes* se consigue mejor tiempo dejando al compilador buscar el tamaño óptimo. En la figura (4.6), vemos la evolución de los tiempos en las diferentes imágenes, en este caso solo en la *GPU*. finalmente, en la tabla (4.6), vemos la mejora de rendimiento de este algoritmo, destacar el caso de la *GPU*, ya que es la plataforma que mejores tiempos consigue, sabiendo que gran parte del tiempo está en la transferencia de datos y no solo en la paralelización.

En el caso de *SCLSU*, tabla (4.7), hemos comparado los tiempos de ambas librerías, el mismo patrón se cumple en ambas imágenes. En el caso de *ViennaCl* el tiempo de transferencia es muy elevado, impidiendo una aceleración eficiente, aun con un tiempo en *Device*

Cuprite	Tiempo (s)	Speedup	Transfer	Device
Serie	2,727	-	0,00 %	100 %
OpenCl CPU (64)	0.100	27,27x	51,10 %	48,9 %
OpenCl GPU (256)	0.060	**45.45x**	**81,80 %**	18,20 %
OpenCl Acelerador (16)	0.138	19.76x	41,80 %	52,20 %
Sintética	Tiempo (s)	Speedup	Transfer	Device
Serie	55,897	-	0,00 %	100 %
OpenCl CPU (64)	0,741	75,43x	30,60 %	69,40 %
OpenCl GPU (1024)	0,374	**149,45x**	**62,70 %**	37,30 %
OpenCl Acelerador (16)	1,118	49.99x	20,60 %	79,40 %

Tabla 4.6: *Tiempos y aceleraciones en diferentes plataformas para el algoritmo SGA.*

muy pequeño (del orden de milisegundos). Por otro lado los tiempos de *ClMagma* son mejores gracias en gran medida a los tiempos de transferencia, son mucho mejores que la otra librería. En cambio los tiempos de ejecución en *Device* son algo peores que *ViennaCl*, no influyen suficientemente como para afirmar que es más rápido la solución a la que llegamos con esta librería. En el caso de utilizar este algoritmo en una plataforma *Aceleradora*, observamos de nuevo el problema de *ViennaCl* con las transferencias, si obviamos este dato se podría conseguir $2x$ de aceleración. Los tiempos obtenidos con este algoritmo usando la librería *ClMagma* están lejos de ser óptimos.

Cuprite	Tiempo (s)	Speedup	Host	Transfer	Device
Serie	0,754	-	0,00 %	0,00 %	100,00 %
ViennaCl GPU	0,409	1,84x	19,80 %	79,40 %	**0,80** %
ClMagma GPU	0,141	5,34x	25,50 %	**59,20** %	15,40 %
ViennaCl Acel.	1,393	-	7,80 %	39,10 %	53,10 %
ClMagma Acel.	113,501	-	0,10 %	0,10 %	99,80 %
Sintetica	Tiempo (s)	Speedup	Host	Transfer	Device
Serie	2,177	-	0,00 %	0,00 %	100,00 %
ViennaCl GPU	1,818	1,19x	21,30 %	78,60 %	**0,10** %
ClMagma GPU	0,212	10,26x	45,10 %	**42,20** %	12,70 %
ViennaCl Acel.	3,948	-	13,50 %	63,80 %	22,70 %
ClMagma Acel.	120,939	-	0,30 %	0,30 %	99,40 %

Tabla 4.7: *Tiempos y aceleraciones en diferentes plataformas para el algoritmo SCLSU.*

A continuación tenemos la cadena de desmezclado formada por *GENE+SCLSU*, como se ha explicado anteriormente el algoritmo *GENE* no solo busca la cantidad de endmembers presentes en la imagen hiperespectral sino que ademas extrae las firmas espectrales. Con estas firmas espectrales podemos crear un mapa de abundancias con el algoritmo *SCLSU*. Como podemos observar en la tabla (4.8), obtenemos una aceleración discreta, teniendo en cuenta que la mayor parte del algoritmo ya usaba la librería *Blas*, la cual supone código optimizado. Lo interesante es ver que la mayor parte del cómputo se encuentra en el primer algoritmo, en caso de que en un futuro se necesite mejorar tiempos sería interesante optimizar este algoritmo, ya que la creación de abundancias lleva menos peso en esta cadena. También destacar como con una imagen pequeña, *Cuprite*, tenemos una aceleración $4,14x$ cuando pasamos a una de mayor tamaño con más endmembers obtenemos una mejora significativa llegando a los $7,63x$.

Cuprite	Tiempo (s)	Speedup	Host	Transfer	Device	GENE	SCLSU
Serie	3,833	-	0,00 %	0,00 %	100,00 %	78,09 %	21,91 %
OpenCl GPU	0,924	4,14x	13,90 %	5,70 %	80,40 %	95,50 %	4,50 %
Sintética	Tiempo (s)	Speedup	Host	Transfer	Device	GENE	SCLSU
Serie	29,772	-	0,00 %	0,00 %	100,00 %	92,68 %	7,32 %
OpenCl GPU	3,901	7,63x	4,10 %	4,40 %	91,50 %	95,20 %	4,80 %

Tabla 4.8: *Tiempos y aceleraciones en GPU para los algoritmos GENE+SCLSU.*

Para finalizar la cadena de desmezclado completa, GENE+SGA+SCLSU, tabla (4.9). Lo primero que observamos es que los algoritmos es que *SGA* tienen un peso mucho mayor que el resto en *serie*, una vez acelerado el peso recae prácticamente en *GENE*. Como se ha explicado anteriormente se ha conseguido una mejora muy sustancial en *SGA*, paralelizando el calculo del determinante y reduciendo el número de cálculos. Por otro lado observamos que para las tres imágenes obtenemos tiempos por debajo de tiempo-real, teniendo en cuenta que tanto para *Cuprite* como para *SubsetWTC* se han calculado 27 y 42 endmembers respectivamente, más de los que contiene las imágenes originales, esto supone que los tiempos serán mejores cuando el algoritmo *GENE* obtenga un número de endmembers mas cercano a la realidad

(las imágenes sintéticas obtiene el numero exacto de endmembers).

Cuprite	Tiempo (s)	Speedup	Host	Transfer	Device	GENE	SGA	SCLSU
serie	13,285	-	0,00 %	0,00 %	100,00 %	21,31 %	71,15 %	7,55 %
OpenCl CPU	50,217	-	0,30 %	0,60 %	**99,00** %	85,06 %	**0,01** %	14,93 %
OpenCl GPU	1,176	**11,29**x	15,10 %	13,50 %	71,40 %	72,70 %	15,43 %	11,87 %
OpenCl Acel.	897,001	-	0,05 %	0,05 %	**99,90** %	87,42 %	**0,01** %	12,57 %
SubsetWTC	Tiempo (s)	Speedup	Host	Transfer	Device	GENE	SGA	SCLSU
serie	116,072	-	0,00 %	0,00 %	100,00 %	18,98 %	79,30 %	1.72 %
OpenCl CPU	55,388	2,09x	0,80 %	1,60 %	97,60 %	84,34 %	3,19 %	12,47 %
OpenCl GPU	4,004	**29,01**x	8,50 %	12,50 %	79,00 %	69,88 %	18,70 %	11,42 %
OpenCl Acel.	947,023	-	0,00 %	0,10 %	**99,90** %	86,56 %	**0.01** %	13,43 %
Sintética	Tiempo (s)	Speedup	Host	Transfer	Device	GENE	SGA	SCLSU
serie	85,669	-	0,00 %	0,00 %	100,00 %	32,21 %	65,24 %	2,55 %
OpenCl CPU	49,573	1,72x	1,3 %	2,6 %	96,1 %	85,20 %	4,88 %	9,91 %
OpenCl GPU	5,175	**16,55**x	9,50 %	14,30 %	76,20 %	72,48 %	15,07 %	12,45 %
OpenCl Acel.	911,259	-	0,10 %	0,20 %	99,70 %	86,24 %	**0,01** %	13,75 %

Tabla 4.9: *Tiempos y aceleraciones en GPU para los algoritmos GENE+SGA+SCLSU.*

Se ha estudiado el comportamiento de las funciones *clMagma* más utilizadas, **magma_dgemm**, y se ha comprobado que ante un aumento significativo en el tamaño de las matrices, se mejoran los tiempos en mayor medida, es decir que amortizamos mejor el gasto que se hace al transferir la imagen a *Device* y su procesamiento. Por todo ello si nos enfrentamos a imágenes de mayor tamaño, en la web de AVIRIS[11] podemos encontrar imágenes hiperespectrales de hasta 10GB, sabemos de ante mano que podemos obtener unos resultados suficientemente buenos para pasar el corte de tiempo-real. Destacar que ambas librerías aun están en una etapa temprana de desarrollo, es posible que sigan mejorando y den mejor soporte a otras plataformas.

[11] http://aviris.jpl.nasa.gov/alt_locator/

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El desarrollo del análisis de imágenes hiperespectrales está aun por despegar, en gran medida debido a la falta de eficiencia en el proceso de desmezclado espectral, donde en el presente trabajo se propone la paralelización de dicho proceso. Para ello se ha utilizado el paradigma de programación de OpenCl, que permite paralelizar algoritmos en una amplia gama de plataformas heterogéneas como pueden ser GPU's, CPU's, Xeon Phi, DSP's o FPGA's.

Las imágenes hiperespectrales tienen muchas utilidades bien sean civiles o militares. Aunque es un campo relativamente nuevo, el hecho de tener tantas utilidades da lugar que se encuentren en constante desarrollo y mejora. Estas imágenes son tomadas por un tipo de sensor llamado espectómetro. Este sensor se encarga de medir la radiación solar reflejada por los diferentes materiales de la Tierra que se encuentran en la muestra tomada. En dichas imágenes podemos encontrar dos tipos de píxeles, puros y mezcla. Los píxeles puros se componen de un solo material, con una firma espectral definida, estos píxeles se conocen como endmembers, mientras que los píxeles mezcla son una combinación de varios endmembers. Es esto lo que se conoce como el problema de la mezcla espectral. La cadena de desmezclado pretende solucionar este problema, para ello propone identificación del número de endmembers, extracción de los mismos y la extracción del mapa de abundancias de cada

endmember.

La cadena de desmezclado propuesta en el presente trabajo trata de probar nuevos algoritmos en este campo, con unos resultados muy positivos. Se ha conseguido reducir el tiempo de ejecución más allá del límite de tiempo-real. Dicho tiempo es crítico en ciertas situaciones, por ejemplo en las descritas por la imagen *SubsetWTC*. Pero es solo un ejemplo, muchos más estudios podrán producirse gracias a los avances en el análisis de las imágenes hiperespectrales, estudios de cambio climático, cambios en los ecosistemas, por citar algunos ejemplos.

Para llevar a cabo la comparativa desde distintos puntos de vista se han utilizado imágenes sintéticas e imágenes reales. Las imágenes sintéticas siguen patrones espaciales similares a los que produce la naturaleza mediante la utilización de fractales y comprenden una detallada simulación mediante el uso de firmas espectrales reales y ruido en diferentes proporciones. Por otro lado las imágenes reales usadas en la comparativa han sido tomadas por el sensor *AVIRIS*.

Se presenta un análisis del estudio realizado al ejecutar la cadena de desmezclado en diferentes dispositivos. Donde se puede comprobar en las tablas mostradas en el capítulo anterior que las *GPUs* son el dispositivo con mayor rendimiento. Los *speedup* son diferentes dependiendo de la imagen hiperespectral a tratar, esto se debe básicamente a su tamaño. La paralelización es un proceso que mejora de forma considerable aquellas partes del código con una gran carga computacional, la diferencia en el *speedup* se debe a que el tiempo que se encuentra ejecutando el algoritmo es mayor. Por esto se puede observar un *speedup* mayor en imágenes de gran tamaño, como *SubsetWTC*. Esto ocurre debido a que en las imágenes hiperespectrales de menor tamaño el tiempo transcurrido en el código paralelizado es muy breve, empleando gran parte del tiempo total en la transferencia de datos. En el proceso de tratamiento se llega a obtener un ***speedup*** de ***29x*** con el uso de una *GPU*, más concretamente una ***NVidia GeForce Gtx 980***.

Partiendo de la base de los datos obtenidos, se puede concluir que en las imágenes de

gran tamaño también se consigue una mejoría del rendimiento con el uso de una *CPU*, en nuestro caso, una ***Intel Xeon E5-2695***. Aunque el rendimiento es bastante discreto, llegando en *Subset WTC* a lograr un ***speedup*** de ***2x***. En la imagen de menor tamaño, la imagen *Cuprite*, no se consigue obtener un mejor tiempo que en su proceso en serie.

El *acelerador* ***Intel Xeon Phi 31s1P*** ha sido el peor parado en este estudio, dado que con ninguna imagen se ha conseguido superar los tiempos del tratamiento en serie. Esto es debido a que la librería utilizada, *ClMagma*, no esta optimizada para su uso en esta plataforma. Esperamos que en un futuro cercano se puedan mejorar los tiempos ya que hemos usado esta plataforma en el algoritmo *SGA* consiguiendo una mejora significativa en los tiempos respecto a *serie*.

5.2. Trabajo futuro

En la actualidad el campo de la investigación hiperespectral está en continuo desarrollo. En gran parte se debe a sus amplias aplicaciones, aunque es evidente que las mejoras en los sensores para tomar estas muestras también son bastante influyentes. Por esto cada vez aparecen nuevos algoritmos tratando de mejorar el procesamiento de las imágenes hiperespectrales, así como soluciones para optimizar los ya existentes.

En un futuro no muy lejano, no sería raro encontrarnos alguna idea revolucionaria que tratase de erradicar los problemas que actualmente conllevan las imágenes hiperespectrales.

Pensamos que sería bueno en un futuro tener la posibilidad de poder implementar diferentes algoritmos de cada etapa del proceso de desmezclado espectral. Con esto podríamos lograr obtener resultados con diferentes combinaciones, y realizar un estudio para determinar que cadena obtiene mejor rendimiento.

Otra de las ideas que nos gustaría poder materializar es la de poder implementar la cadena de desmezclado espectral para plataformas de bajo consumo. Esto no solo reduciría el consumo, sino que nos permitiría poder situar el dispositivo en la plataforma que se ubique

el sensor. Con ello conseguiríamos reducir el tamaño que este tiene que enviar a la tierra, puesto que al procesar los datos en la propia plataforma no sería necesario el envío de la muestra completa. Lo cual supondría una reducción del tiempo de transmisión a la par que de tamaño de datos almacenados.

Una de estas plataformas son las FPGAs, son un dispositivo hardware reconfigurable de bajo consumo, están compuestas por bloques de lógica, los cuales son reprogramables. Esto no solo proporciona la ventaja de que se puedan procesar las imágenes en la propia plataforma, sino que según el tipo de análisis que queramos llevar a cabo nos permite cargar los algoritmos que mejor se adapten a dicho análisis, con el fin de obtener el mejor rendimiento. Para ello deberíamos cambiar parte del código puesto que las FPGAs no soportan la versión 1.2 de opencl (en la cual esta realizado nuestro proyecto), además de la adaptación del mapeado en memoria, puesto que las FPGAs tienen uno específico.

Bibliografía

[1] Chein-I Chang. An information-theoretic approach to spectral variability, similarity, and discrimination for hyperspectral image analysis. *IEEE Transactions on Information Theory*, 46(5):1927–1932, Aug 2000.

[2] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton. Advances in spectral-spatial classification of hyperspectral images. *Proceedings of the IEEE*, 101(3):652–675, March 2013.

[3] L. Zhao, C. I. Chang, S. Y. Chen, C. C. Wu, and M. Fan. Endmember-specified virtual dimensionality in hyperspectral imagery. In *2014 IEEE Geoscience and Remote Sensing Symposium*, pages 3466–3469, July 2014.

[4] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[5] J. C. Harsanyi and C. I. Chang. Hyperspectral image classification and dimensionality reduction: an orthogonal subspace projection approach. *IEEE Transactions on Geoscience and Remote Sensing*, 32(4):779–785, Jul 1994.

[6] T. Lillesand R. Kiefer and J. Chipman. *Remote Sensing and Image Interpretation*. John Wiley and Sons, Inc, 2004.

[7] D. A. Landgrebe. *Signal Theory Methods in Multispectral Remote Sensing*. John Wiley & Sons: New York, 2003.

[8] D. Landgrebe. Hyperspectral image data analysis. 19:17–28, 2002.

[9] C.-I Chang. *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Kluwer Academic/Plenum, 2003.

[10] A. F. Goetz and B. Kindel. Comparison of unmixing result derived from aviris, high and low resolution, and hydice images at cuprite, nv. *Summaries of the Fifth Annual JPL Airborne Earth Science Workshop*, 1:23–26, 1999.

[11] R. O. Green and B. Pavri. Aviris in-flight calibration experiment, sensitivity analysis, and intraflight stability. *JPL AVIRIS Workshop*, 2000.

[12] M. O. Smith J. B. Adams and P. E. Johnson. Spectral mixture modeling: A new analysis of rock and soil types at the viking lander 1 site. *Journal of Geophysical Research*, 91:8098–8112, 1986.

[13] D. E. Sabol, J. B. Adams, and M. O. Smith. Predicting the spectral detectability of surface materials using spectral mixture analysis. In *Geoscience and Remote Sensing Symposium, 1990. IGARSS '90. 'Remote Sensing Science for the Nineties'., 10th Annual International*, pages 967–970, May 1990.

[14] N. Keshava and J. F. Mustard. Spectral unmixing. *IEEE Signal Processing Magazine*, 19(1):44–57, 2002.

[15] M. Petrou and P. G. Foschi. Confidence in linear spectral unmixing of sinle pixels. *IEEE Trans. Geosci. and Remote Sens*, 37:624–626, 1999.

[16] M. Xu, B. Du, and L. Zhang. Spatial-spectral information based abundance-constrained endmember extraction methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):2004–2015, June 2014.

[17] L. M. Bruce J. E. Ball and N. Younan. Hyperspectral pixel unmixing via spectral band selection and dc-insensitive singular value decomposition. *IEEE Geoscience and Remote Sensing Letters*, 4(3):382–386, 2007.

[18] J. Bioucas-Dias X. Jia A. Plaza, Q. Du and F. Kruse. Foreword to the special issue

on spectral unmixing of remotely sensed data. *IEEE Trans. Geosci. and Remote Sens*, 49(11):4103–4110, 2011.

[19] Chein-I Chang and D. C. Heinz. Constrained subpixel target detection for remotely sensed imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 38(3):1144–1159, May 2000.

[20] R. Heylen, P. Scheunders, A. Rangarajan, and P. Gader. Nonlinear unmixing by using different metrics in a linear unmixing chain. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2655–2664, June 2015.

[21] R. N. Clark and T. L. Roush. Reflectance spectroscopy: Quantitative analysis techniques for remote sensing applications. 89(7):6329–6340, 1984.

[22] Y. Altmann, M. Pereyra, and S. McLaughlin. Nonlinear spectral unmixing using residual component analysis and a gamma markov random field. In *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2015 IEEE 6th International Workshop on*, pages 165–168, Dec 2015.

[23] Min Choi, DaeWoo Lee, and SeungRyoul Maeng. Cluster computing environment supporting single system image. In *Cluster Computing, 2004 IEEE International Conference on*, pages 235–243, Sept 2004.

[24] K. Koonsanit and C. Jaruskulchai. A simple framework of segmentation for hyperspectral images using clustering techniques. In *SICE Annual Conference (SICE), 2011 Proceedings of*, pages 43–47, Sept 2011.

[25] C. Lazar, L. Demarchi, D. Steenhoff, J. C. W. Chan, A. Nowé, and H. Sahli. Local linear spectral unmixing via cluster analysis and non-negative matrix factorization for hyperspectral (chris/proba) imagery. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 7267–7270, July 2012.

[26] C. I. Chang and Y. Li. Recursive band processing of automatic target generation process for finding unsupervised targets in hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, PP(99):1–14, 2016.

[27] C. I. Chang, C. C. Wu, W. Liu, and Y. C. Ouyang. A new growing method for simplex-based endmember extraction algorithm. *IEEE Transactions on Geoscience and Remote Sensing*, 44(10):2804–2819, Oct 2006.

[28] M. Maghrbay, R. Ammar, and S. Rajasekaran. Improving n-finder technique for extracting endmembers. In *2011 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 042–049, Dec 2011.

[29] *OpenCL^{TM} Optimization Case Study: Simple Reductions*. http://developer.amd.com/resources/documentation-articles/articles-whitepapers/opencl-optimization-case-study-simple-reductions/.

Apéndice A

Introduction

Technological devices are upgrading fast, thats makes the software around them, also be upgraded, in order to exploit the full potencial of the new features that come with them.

In this proyect we test this topic, how the technology is advancing and how can we improve previous works with the new technology features. More specificaly about image processing taken from remote sensors, thats so called **Hyperespectral images**. If we compare these images with a normal digital image, we can see that they are related. The main diference is each of every pixel is formed from a set of different wave lenghts and in a digital image each pixel is made from a three different wave lenghts (red, green, blue).

Remote sensor are on board of satellites or flying platforms, and takes images of the Earth's surface. Sensors measure the reflected radiation of diferent materials each from a different wave lenght, thats so called espectral signature.

Spectral image analysis has many purposes, a few of them are: check pollution from water or air, fires where is the hot spot and how is evolving, identify different species in an ecosystem, purity analysis of ground materials and last but not least military aplications, detect ground mines or locate targets.

The technique we are working with is called spectral unmixing, this technique allow us to obtain spectral signatures of the materials present on the image. At this point we can differenciate pure mixture pixel, called endmember, and mixed pixel, which is formed for more than one pure spectral signature.

Main problem of this technique is the time it takes to process, even small images takes way too long, in some approaches are using large scale Clusters, but this do not look the right way to solve the problem.

In this work we propose an alternative way to solve this issue, in the lastest years, leisure time has grown as a consecuence of the game industry. Developing new and more capable Graphics Process Units, known as **GPU**'s, as well as Central Process Units, **CPU**'s. we are going to take advantage of this graphics units.

These kind of hardware it is the best solution around problems that can be parallelize, such as image processing because all the data are independent from each other.

The main goal of the present proyect is speed up spectral unmixing process for hyperespectral images, solving all the issues around it, as the main goal is keep processing time below real-time limit. Current solutions are not viable, even small images are time consuming, and can not achieve real-time. The purpose of this proyect will be, development of the unmixing chain in parallela way, in order to achieve real-time processing, with programing paradigm of OpenCl.

Unmixing chain is formed by three algorithms, each of them has been developed a diferent way to parallelize it.

In first place, Geometry-based Estimation of number of endmembers, known as GENE, this algorithm have as goal, to detect the number of the endmembers presents in the current image. For this algorithm has been used a hybrid speed up, half of it using ClMagma Library and half developed by ourselves inside a OpenCl Kernel.

Followed by, Simplex Growing algorithm, SGA, this algorithm has the goal of extract all the endmembers, along with its bands, that have been found by the previous algorithm. In this case only OpenCl kernel has been used as speed up process.

Last but not least, Sum to one Constrained Linear Spectral Umnixing, SCLSU, this algorithm have the purpose of make abundances maps of each endmember found for the last algorithm. Here we tested two different kind of libraries, in order to check which one give

us the fastest solution.

Since OpenCl it is a unified programming model for accelerating algorithms on heterogeneous systems, it is possible to test in which platform can achieve the best performance. In this proyect has been used Intel Xeon E5-2695 v3 as CPU Platform, NVidia GeForce GTX 980 as GPU and Intel Xeon Phi 31S1P as Co-procesor.

Apéndice B

Conclusion

B.1. Conclusions

The development of hyperspectral image analysis is yet to lift off, thats because the lack of efficiency in the spectral unmixing analysis. In the present paper it is proposed the parallelization of this process. For this purpose, it has been used OpenCl programming paradigm, that enables parallelize algorithms in a wide range of heterogeneous platforms such as GPU's , CPU's , Xeon Phi , DSP's or FPGA's

Spectral unmixing chain proposed in this paper is testing a new kind of algorithms in this field, with a very positive resuls. It is achieved to reduce the execution time beyond the real-time limit. This time is critical in certain situations, for example those decribed in the *SubsetWTC* image. But thats just an example, many more studies have yet to come thanks to the advance in the hiperespectral images analysis. Naming few of them, studies of climate change or changes in ecosystems.

In this paper it is present an analysis after the execution of unmixing chain on different platforms. In the tables of the previous chapter, we can see that *GPU* have the hightest performance. Speedup results are different depending on the hyperespectral image that we are working on, thats because its size. Parallelization is a process that significantly enhances those parts of the code with a large computational load. Thats why we can see the best performance in *SubsetWTC* image. In this process the best performance its up to ***29x***

Speedup, using GPU (Nvidia Geforce Gtx 980).

From the data extracted, we can conclude the bigger the image size it is, the better performance in the *CPU* we get, in our case ***Intel Xeon E5-2695*** CPU. Although the performance is quite discreet, the bigger image gets 2x speedup. In small size images like *Cuprite* we can not achieve Speedup.

Accelerators have been the worse performance we get, this is because the library we used, *ClMagma*, this library is not optimized for this kind of platforms. We look forward in near future that can get better speedup, since we have no problems in SGA algorithm using accelerator as platform.

B.2. Lines of future work

Nowadays the researching field of hyperspectral images is in steady development. Mostly because of its wide range of applications, although it is taken for granted that the improvement in the sensors which take those samples are very influential too. Due this fact, new algorithms emerge trying to improve the processing of hyperspectral images, as well as answers about how to improve those algorithms who already exist.

In a no so distant future, it wouldn't be weird to find out about a new innovative idea that will try to solve the current problems of hyperspectral images.

In the future we would like to have the chance to implement different algorithms of each stage of the process of spectral descrambling. So we could obtain better results with differents schemes and perform several studies in order to define which chains obtains a better performance.

Another idea we would like to implement is the chain of spectral descrambling for energy-saving platforms. This will not only lower the energy consumption but will also let us position the platforms where the sensor is located. With all this in mind we could reduce the size data the platforms have to send to earth, because processing data in the platform makes sending the full sample size no longer neccesary. This will also mean a reduction of transmision times

and the size of the stored data.

One of this platforms is the FPGAs, its a energy-saving hardware device, composed of reprogramable logic blocks. This not only provides the advantage of processing the images directly in the platform , but also depending on the tests we want to carry on , we could load the algorithms that better suit to such testing, so we can obtain better performance of the FPGA. In order to achieve this we have to change our code because FPGAs cant endure opencl 1.2 (which is the version we used in our project), neither the adaptation of memory mapping, because FPGA's have their own adaptation.

Apéndice C

Reparto de trabajo

C.1. Orueta Moreno, Carlos

La primera reunión que tuvimos con nuestros directores de proyecto fue a finales de Junio, donde nos hablaron acerca del proyecto, los términos generales del mismo. Tras esta reunión ya nos decidimos por este proyecto, ya que nos parecía muy interesante e innovadora la materia, y un campo en continuos avances.

En la siguiente reunión, ya en Julio, nos proporcionaron documentación, y el primer algoritmo a paralelizar, el algoritmo *SGA*. A lo largo del verano me documenté sobre las imágenes hiperespectrales, ya que para poder empezar con el proyecto, necesitaba tener una base consolidada de lo que ibamos a tratar. Una vez tuve unas bases acerca de las imágenes, era el momento de ponerse con el algoritmo. Éste estaba escrito en código *matlab*, a pesar de que lo habíamos visto en una asigntura, tanto mi compañero como yo tuvimos que ampliar los conocimientos de dicho lenguaje para poder comprender el algoritmo para su futuro tratamiento.

Cuando terminó el verano tuvimos otra reunión en la cual presentamos el código del algoritmo *SGA* en lenguaje *C*, después de esta reunión se propusieron objetivos a corto plazo. Estos objetivos consistían en paralelizar este algoritmo con el uso del lenguaje *Opencl*. Tuve que refrescar los conocimientos en *Opencl*, este lenguaje lo habíamos utilizado ya en una optativa. Con los conocimientos refrescados, comence con la paralelización. La primera im-

plementación en opencl propuso algunas dificultades, como fue el cálculo del determinante, personalmente yo tuve muchos problemas con la reducción, dado que no terminaba de comprenderla. Finalmente, tras documentarme en diferentes páginas de internet, foros dedicados a *opencl* y largas conversaciones con mi compañero, conseguí tener una reducción, aunque esta fuera muy simple. En navidades entregamos el código del algoritmo *SGA*, el cual iba a ser enviado a un congreso. Este fue uno de los objetivos parciales del proyecto, lograr tener resultados funcionales capaces de tener un buen rendimiento con el fin de presentarlo en este congreso (explicado en el apartado de publicaiones).

En la reunión de diciembre, en la que presentamos el algoritmo *SGA* paralelizado, se nos propuso el siguiente algoritmo de la cadena. El algoritmo *SCLSU* encargado de la estimación de abundancias, para este caso ibamos a hacer uso de librerias basadas en *Opencl*, mas en concreto *ViennaCL* y *clMAGMA*. Este algoritmo estaba escrito en lenguaje *c* y ya hacía uso de librerias, de *BLAS*. En navidades estuve documentandome acerca de dichas librerias y realizando pruebas para comprobar el funcionamiento de las mismas.

Después de finalizar los exámenes de febrero, retomé el desarrollo del algoritmo *SCLSU*, como mi compañero habia desarrollado el algoritmo en *ViennaCL*, pasé a la implementación en *clMAGMA*, encontre ciertas dificultades en las reservas de memoria. Tras consultar ejemplos de *clMAGMA* fuí capaz de solventar.

En otra reunión se abordó el rendimiento de este algoritmo, que era muy pobre en imágenes pequeñas, con lo cual se nos proporcionó otra implementación en código *C* del mismo.

Por último, para comlpetar la cadena de desmezclado nos quedaba un algoritmo, el algoritmo *GENE*, el de estimación de número de endmembers en la imagen hiperespectral. Se nos proporcionó una implementación en serie y otra en paralelo, la versión en paralelo estaba implementada en *CUDA* y haciendo uso de librerias, con el fin de que pudieramos realizar comparaciones de rendimiento. En este caso, el objetivo era implementar el algoritmo mezclando el lenguaje *Opencl* y la libreria *clMagma*. La mayor dificultad que nos surgió en esta

paralelización fue la de reservar en memoria variables capaces de poder ser utilizadas tanto por los kernel como por las librerias, es decir que tuvieran un contexto común. Se optaron por dos opciones: implementar la función de las librerias en *Opencl* o crear un contexto común. Aquí yo me encargué de crear el kernel, mientras mi compañero se encargaba de crear el contexto común. Finalmente, en un foro nos proporcionaron una solución con contexto común, la cual obtenía mejor rendimiento. Asi que optamos por esa solución.

Ya solo nos quedaba la memoria, la cual redactamos a medias, revisando las partes escritas por el otro. Consiguiendo así corregirnos las faltas ortográficas o problemas de redacción que cometieramos.

C.2. Rodríguez Navarro, Miguel

En esta sección voy a explicar las tareas que he realizado durante el desarrollo del proyecto.

Durante la primera reunión, a principios de Julio 2015, se nos entregaron documentación de lo que seria el proyecto, es decir empezamos ya en verano a trabajar sobre la materia, para tener una base solida de que es lo que íbamos a realizar y empezar a pensar formas de realizarlo. En principio durante el verano solo se realizaron estudios de documentación, lectura de una tesis para aprender los conceptos básicos.

En primer lugar se nos entregaron dos algoritmos, el primero (**SGA**) estaba escrito en lenguaje de programación de **Matlab**. Particularmente yo nunca había tocado este lenguaje así que hubo que mirar documentación antes de ponerse a traducir este algoritmo. Es bastante intuitivo después de consultar varios ejemplos, se pudo empezar a traducir primero a lenguaje C y posteriormente a OpenCl. Uno de los primeros escollos que me encontré es el calculo de determinantes de una matriz de tamaño variable en la que el tamaño máximo sería el numero de endmembers. Por tanto tuve que buscar información ya que *Matlab* utiliza funciones que no tengo acceso a su código fuente. Encontré que la mejor forma de atacar el problema es con la descomposición LU (explicada en el apartado 3.2.2).

Un vez traducido el código a C, el siguiente paso fue rápido, la traducción a OpenCl era bastante sencilla. Ya teníamos experiencia programando *OpenCl*, pero no utilizando *memoria local* así que en la primera versión no se utilizo memoria local. Con una mejora en los tiempos entorno a 6x.

En una de las siguientes reuniones, en torno a diciembre 2015, entregamos a los profesores el primer código OpenCl de SGA para su revisión, entonces se nos encomendó empezar a mirar el uso de las librerías, *ViennaCl* y *ClMagma*, ya que serian estas las que utilizaríamos para traducir el segundo algoritmo, **SCLSU**. Con una documentación muy extensa pero de escasos ejemplos prácticos, la única forma de de aprender fue probando. La primera librería que probé fue *ViennaCl* ya que me resultó más sencillo a la hora de ejecutarlo. Hasta el

momento no tenia mucha experiencia creando *Makefiles* así que también tuve que buscar información al respecto. Uno de los problemas que surgieron entonces fue que no coincidían los resultados respecto al código original. El algoritmo que teníamos que traducir estaba escrito en **C** y usaba la librería *Blas*. Algo que no esta del todo claramente explicado y que tarde bastante en darme cuenta fue que la librería *Blas* utilizaba ordenación por columnas mientras que *ViennaCl* utiliza ordenación por filas, por eso no coincidían los resultados.

Pasadas vacaciones de navidad en la siguiente reunión los profesores nos comentaron que podíamos mejorar el algoritmo de *SGA* usando memoria local. En este campo solo se puede abordar con pruebas de ensayo y error, tuve la suerte de encontrar una solución que usando memoria local, paralelizaba mucho el código. También encontré una librería llamada *OpenCv* que me ayudo bastante a la hora de visualizar los resultados.

Al mismo tiempo seguía el desarrollo del algoritmo *SCLSU*, fue entonces cuando se encontró que no se podía mejorar el algoritmo original con las librerías propuestas ya que las matrices que manejaba eran demasiado pequeñas. Se nos entrego un nuevo algoritmo que abordaba el mismo problema pero desde una óptica diferente. Como ya tenia bastante experiencia en ambas librerías la traducción fue bastante rápida.

Cuando ya tuvimos ambos algoritmos paralelizados, se nos propuso completar la cadena de desmezclado incorporando un tercer algoritmo, de esta forma quedaría el trabajo más completo. El problema más complicado con el que tuve que lidiar fue crear un *contexto* común a la librería *ClMagma* y a un kernel *OpenCl*

Esto es solo un ligero esbozo de todas las horas que le he dedicado a este proyecto, pero creo que el resultado a merecido la pena y hemos aprendido mucho.

Apéndice D

Publicaciones

A continuación se van a listar las publicaciones surgidas a raíz del presente trabajo en congresos internacionales y revistas especializadas.

Artículos publicados en congresos internacionales:

- Sergio Bernabé, Guillermo Botella, Jose M. R. Navarro, Carlos Orueta, Manuel Prieto-Matías y Antonio Plaza. "*Parallel implementation of the Simplex Growing Algorithm for hyperespectral unmixing using OpenCl*". IEEE Geoscience and Remote Sensing Symposium (IGARSS'16). Articulo aceptado, pendiente para su publicación.

- Sergio Bernabé, Guillermo Botella, Carlos Orueta, José M. R. Navarro, Manuel Prieto-Matias y Antonio Plaza. "*OpenCL-library-based Implementation of SCLSU Algorithm for Remotely Sensed Hyperspectral Data Exploitation: ClMagma versus ViennaCL*". SPIE Conference on High-Performance Computing in Remote Sensing 2016. Articulo aceptado, pendiente para su publicación.

Publicaciones en revistas con índice de impacto:

- Sergio Bernabé, Guillermo Botella, José M. R. Navarro, Carlos Orueta, Francisco D. Igual, Manuel Prieto-Matias y Antonio Plaza. "*Performance-Power-Energy Evaluation of an OpenCL Implementation of the Simplex Growing Algorithm for Hyperspectral Unmixing*". IEEE Geoscience and Remote Sensing Letters. Artículo enviado. [JCR (2014) = 2.095].